```
**********************************************************
   22227 Fri Dec 21 14:59:53 2018
new/exception_lists/copyright
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
# Copyright (c) 2009, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2011 by Delphix. All rights reserved.
# Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
# Copyright (c) 2018, Joyent, Inc.
#

syntax: glob
exception_lists/closed-bins
exception_lists/copyright
exception_lists/cstyle
exception_lists/hdrchk
usr/src/boot/*
usr/src/cmd/acpi/acpidump/acpidump.h
usr/src/cmd/acpi/acpidump/apdump.c
usr/src/cmd/acpi/acpidump/apfiles.c
usr/src/cmd/acpi/acpidump/apmain.c
usr/src/cmd/acpi/acpidump/osillumostbl.c
usr/src/cmd/acpi/acpidump/osunixdir.c
usr/src/cmd/acpi/acpidump/tbprint.c
usr/src/cmd/acpi/acpidump/tbxfroot.c
usr/src/cmd/acpi/acpidump/utbuffer.c
usr/src/cmd/acpi/acpixtract/acpixtract.c
usr/src/cmd/acpi/acpixtract/acpixtract.h
usr/src/cmd/acpi/acpixtract/axmain.c
usr/src/cmd/acpi/acpixtract/axmain.h
usr/src/cmd/acpi/acpixtract/axutils.c
usr/src/cmd/acpi/acpixtract/axutils.h
usr/src/cmd/acpi/common/getopt.c
usr/src/cmd/acpi/common/utascii.c
usr/src/cmd/acpi/common/utdebug.c
usr/src/cmd/acpi/common/utexcep.c
usr/src/cmd/acpi/common/utglobal.c
usr/src/cmd/acpi/common/utmath.c
usr/src/cmd/acpi/common/utnonansi.c
usr/src/cmd/acpi/common/utprint.c
usr/src/cmd/acpi/common/utxferror.c
usr/src/cmd/cmd-inet/usr.bin/dns-sd/ClientCommon.c
```



```
usr/src/cmd/cmd-inet/usr.bin/dns-sd/ClientCommon.h
usr/src/cmd/cmd-inet/usr.bin/dns-sd/dns-sd.c
usr/src/cmd/cmd-inet/usr.lib/mdnsd/THIRDPARTYLICENSE
usr/src/cmd/cmd-inet/usr.lib/mdnsd/*.[ch]
usr/src/cmd/dtrace/test/tst/common/*/*.out
usr/src/cmd/krb5/kadmin/cli/kadmin_ct.c
usr/src/cmd/krb5/kadmin/cli/kadmin.h
usr/src/cmd/krb5/kadmin/cli/ss_wrapper.c
usr/src/cmd/krb5/kadmin/dbutil/nstrtok.h
usr/src/cmd/krb5/kadmin/dbutil/ovload.c
usr/src/cmd/krb5/kadmin/dbutil/string_table.c
usr/src/cmd/krb5/kadmin/dbutil/strtok.c
usr/src/cmd/krb5/kadmin/dbutil/util.c
usr/src/cmd/krb5/kadmin/kpasswd/kpasswd_strings.h
usr/src/cmd/krb5/kadmin/kpasswd/kpasswd.h
usr/src/cmd/krb5/kadmin/ktutil/ktutil.h
usr/src/cmd/krb5/kadmin/server/server_glue_v1.c
usr/src/cmd/krb5/krb5kdc/extern.c
usr/src/cmd/krb5/krb5kdc/kdc_util.c
usr/src/cmd/krb5/krb5kdc/policy.c
usr/src/cmd/krb5/krb5kdc/replay.c
usr/src/cmd/krb5/krb5kdc/sock2p.c
usr/src/cmd/krb5/ldap_util/kdb5_ldap_list.c
usr/src/cmd/krb5/ldap_util/kdb5_ldap_list.h
usr/src/cmd/krb5/ldap_util/kdb5_ldap_policy.h
usr/src/cmd/krb5/ldap_util/kdb5_ldap_realm.h
usr/src/cmd/krb5/ldap_util/kdb5_ldap_services.h
usr/src/cmd/localedef/data/manual-input.UTF-8
usr/src/cmd/smbsrv/smbd/eventlog.dll
usr/src/cmd/terminfo/termcap.src
usr/src/cmd/terminfo/terminfo.src
usr/src/common/bzip2/LICENSE
usr/src/common/bzip2/Solaris.README.txt
usr/src/common/bzip2/bzlib.h
usr/src/common/bzip2/crctable.c
usr/src/common/bzip2/randtable.c
usr/src/common/bzip2/blocksort.c
usr/src/common/bzip2/compress.c
usr/src/common/bzip2/bzlib.c
usr/src/common/bzip2/decompress.c
usr/src/common/bzip2/bzlib_private.h
usr/src/common/bzip2/huffman.c
usr/src/common/ficl/*
usr/src/data/hwdata/THIRDPARTYLICENSE.efifixes.descrip
usr/src/data/hwdata/THIRDPARTYLICENSE.efifixes.tmpl
usr/src/grub/grub-0.97/stage2/Makefile.am
usr/src/grub/grub-0.97/stage2/builtins.c
usr/src/grub/grub-0.97/stage2/disk_io.c
usr/src/grub/grub-0.97/stage2/pc_slice.h
usr/src/grub/grub-0.97/stage2/gpt.h
usr/src/grub/grub-0.97/stage2/shared.h
usr/src/head/rpcsvc/THIRDPARTYLICENSE.nfs4_prot.descrip
usr/src/lib/gss_mechs/mech_krb5/crypto/cksumtype_to_string.c
usr/src/lib/gss_mechs/mech_krb5/crypto/coll_proof_cksum.c
usr/src/lib/gss_mechs/mech_krb5/crypto/enctype_compare.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keyed_checksum_types.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keyed_cksum.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keylengths.c
usr/src/lib/gss_mechs/mech_krb5/crypto/old/des_stringtokey.c
usr/src/lib/gss_mechs/mech_krb5/crypto/random_to_key.c
usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_cksumtype.c
usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_enctype.c
usr/src/lib/gss_mechs/mech_krb5/crypto/valid_cksumtype.c
usr/src/lib/gss_mechs/mech_krb5/et/error_table.h
usr/src/lib/gss_mechs/mech_krb5/et/internal.h
usr/src/lib/gss_mechs/mech_krb5/et/mit-sipb-copyright.h
```

usr/src/lib/gss_mechs/mech_krb5/include/cache-addrinfo.h
usr/src/lib/gss_mechs/mech_krb5/include/cm.h
usr/src/lib/gss_mechs/mech_krb5/include/db-config.h
usr/src/lib/gss_mechs/mech_krb5/include/db.h
usr/src/lib/gss_mechs/mech_krb5/include/fake-addrinfo.h
usr/src/lib/gss_mechs/mech_krb5/include/foreachaddr.h
usr/src/lib/gss_mechs/mech_krb5/include/k5-int-pkinit.h
usr/src/lib/gss_mechs/mech_krb5/include/k5-utf8.h
usr/src/lib/gss_mechs/mech_krb5/include/kdb_kt.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5_libinit.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_defs.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_proto.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/copyright.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-err.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-plugin.h
usr/src/lib/gss_mechs/mech_krb5/include/locate_plugin.h
usr/src/lib/gss_mechs/mech_krb5/include/port-sockets.h
usr/src/lib/gss_mechs/mech_krb5/include/preauth_plugin.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1buf.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krbasn1.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/ldap_key_seq.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cccopy.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefops.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/fcc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/scc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ser_cc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/error_tables/adm_err.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktadd.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktremove.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/read_servi.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_comp.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_order.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_srch.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/appdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_pr_ext.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_princ.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/chk_trans.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cleanup.h
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_addrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_creds.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_data.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_tick.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decode_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decrypt_tk.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/deltat.c



usr/src/lib/gss_mechs/mech_krb5/krb5/krb/enc_helper.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encode_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encrypt_tk.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/free_rtree.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gc_via_tkt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gen_subkey.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_opt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/init_keyblock.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/int-proto.h
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kdc_rep_dc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kerrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_error.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_rep.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/pr_to_salt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/preauth.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_error.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_rep.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_req.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/tgtname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/valid_times.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/vic_opt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/walk_rtree.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/accessor.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/changepw.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnssrv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_hstrl.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_krbhs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/full_ipadr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_port.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_rname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/genaddrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/get_krbhst.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gmt_mktime.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/hostaddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/localaddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/lock_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/mk_faddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_read.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_write.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/os-proto.h
usr/src/lib/gss_mechs/mech_krb5/krb5/os/osconfig.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/port2ip.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/prompter.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/promptusr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_msg.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_pwd.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/realm_dom.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/realm_iter.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/thread_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/unlck_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/ustime.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/write_msg.c
usr/src/lib/gss_mechs/mech_krb5/krb5/posix/daemon.c
usr/src/lib/gss_mechs/mech_krb5/krb5/posix/setenv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_base.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_conv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_io.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_none.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rcfns.c
usr/src/lib/gss_mechs/mech_krb5/mech/add_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/compare_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/context_time.c
usr/src/lib/gss_mechs/mech_krb5/mech/copy_ccache.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_com_err_status.c

```
usr/src/lib/gss_mechs/mech_krb5/mech/disp_major_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/export_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/export_sec_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/gss_libinit.h
usr/src/lib/gss_mechs/mech_krb5/mech/import_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/indicate_mechs.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_names.c
usr/src/lib/gss_mechs/mech_krb5/mech/oid_ops.c
usr/src/lib/gss_mechs/mech_krb5/mech/process_context_token.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_oid.c
usr/src/lib/gss_mechs/mech_krb5/mech/set_allowable_enctypes.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_buffer.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_ctxsetup.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_dup.c
usr/src/lib/gss_mechs/mech_krb5/mech/utl_nohash_validate.c
usr/src/lib/gss_mechs/mech_krb5/profile/prof_err.h
usr/src/lib/gss_mechs/mech_krb5/profile/prof_get.c
usr/src/lib/gss_mechs/mech_krb5/profile/prof_set.c
usr/src/lib/gss_mechs/mech_krb5/support/errors.c
usr/src/lib/gss_mechs/mech_krb5/support/fake-addrinfo.c
usr/src/lib/gss_mechs/mech_krb5/support/init-addrinfo.c
usr/src/lib/gss_mechs/mech_krb5/support/supp-int.h
usr/src/lib/gss_mechs/mech_krb5/support/threads.c
usr/src/lib/gss_mechs/mech_krb5/support/utf8.c
usr/src/lib/krb5/dyn/dyn_append.c
usr/src/lib/krb5/dyn/dyn_create.c
usr/src/lib/krb5/dyn/dyn_debug.c
usr/src/lib/krb5/dyn/dyn_delete.c
usr/src/lib/krb5/dyn/dyn_initzero.c
usr/src/lib/krb5/dyn/dyn_insert.c
usr/src/lib/krb5/dyn/dyn_paranoid.c
usr/src/lib/krb5/dyn/dyn_put.c
usr/src/lib/krb5/dyn/dyn_realloc.c
usr/src/lib/krb5/dyn/dyn_size.c
usr/src/lib/krb5/kadm5/admin_internal.h
usr/src/lib/krb5/kadm5/admin_xdr.h
usr/src/lib/krb5/kadm5/chpass_util_strings.h
usr/src/lib/krb5/kadm5/clnt/client_handle.c
usr/src/lib/krb5/kadm5/clnt/clnt_chpass_util.c
usr/src/lib/krb5/kadm5/kadm_rpc.h
usr/src/lib/krb5/kadm5/misc_free.c
usr/src/lib/krb5/kadm5/srv/server_dict.c
usr/src/lib/krb5/kadm5/srv/server_handle.c
usr/src/lib/krb5/kadm5/srv/server_misc.c
usr/src/lib/krb5/kadm5/srv/svr_iters.c
usr/src/lib/krb5/kadm5/srv/svr_misc_free.c
usr/src/lib/krb5/kadm5/srv/svr_policy.c
usr/src/lib/krb5/kadm5/srv/xdr_alloc.c
usr/src/lib/krb5/kdb/adb_err.h
usr/src/lib/krb5/kdb/kdb5.h
usr/src/lib/krb5/kdb/keytab.c
usr/src/lib/krb5/plugins/kdb/db2/adb_policy.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.h
usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_close.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_conv.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_delete.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_get.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_open.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_overflow.c
```



```
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_page.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_put.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_search.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_split.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_utils.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/db/db.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_bigkey.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_func.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_log2.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_page.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hsearch.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-ndbm.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_close.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_delete.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_get.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_open.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_put.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_search.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_seq.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_utils.c
usr/src/lib/krb5/plugins/kdb/db2/pol_xdr.c
usr/src/lib/krb5/plugins/kdb/db2/policy_db.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_fetch_mkey.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_rights.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_lib.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_profile.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_srv.c
usr/src/lib/krb5/ss/copyright.h
usr/src/lib/krb5/ss/mit-sipb-copyright.h
usr/src/lib/krb5/ss/options.c
usr/src/lib/krb5/ss/std_rqs.c
usr/src/lib/krb5/ss/utils.c
usr/src/lib/libdns_sd/THIRDPARTYLICENSE
usr/src/lib/libdns_sd/common/*.[ch]
usr/src/lib/libdns_sd/java/common/JNISupport.c
usr/src/lib/libdns_sd/java/com/apple/dnssd/*.java
usr/src/lib/libdns_sd/java/com/apple/dnssd/docs/examples/src/*
usr/src/lib/librstp/common/*.[ch]
usr/src/lib/librstp/common/[CRT]*
# these have copyrights that the nit checker doesn't grok
usr/src/lib/libsmbfs/netsmb/spnego.h
usr/src/lib/libsmbfs/smb/derparse.[ch]
usr/src/lib/libsmbfs/smb/spnego.c
usr/src/lib/libsmbfs/smb/spnegoparse.[ch]
usr/src/test/util-tests/tests/dis/*/*.out
usr/src/test/util-tests/tests/grep_xpg4/files/gout*
```

```
usr/src/test/util-tests/tests/grep_xpg4/files/test*
usr/src/test/util-tests/tests/libsff/*.out
usr/src/test/zfs-tests/tests/functional/history/*Z
usr/src/test/zfs-tests/tests/functional/history/*txt
usr/src/tools/btxld/btx.h
usr/src/tools/btxld/btxld.8
usr/src/tools/btxld/btxld.c
usr/src/tools/btxld/elfh.c
usr/src/tools/btxld/elfh.h
usr/src/tools/btxld/imgact_aout.h
usr/src/tools/smatch/src/*
usr/src/uts/intel/nsmb/ioc_check.ref
usr/src/uts/intel/os/splashimage.xpm
usr/src/uts/common/gssapi/mechs/krb5/crypto/block_size.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/checksum_length.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/f_parity.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/weak_key.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/encrypt_length.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_crc32.c
usr/src/uts/common/gssapi/mechs/krb5/include/aes_s2k.h
usr/src/uts/common/gssapi/mechs/krb5/include/auth_con.h
usr/src/uts/common/gssapi/mechs/krb5/include/cksumtypes.h
usr/src/uts/common/gssapi/mechs/krb5/include/crc-32.h
usr/src/uts/common/gssapi/mechs/krb5/include/dk.h
usr/src/uts/common/gssapi/mechs/krb5/include/enc_provider.h
usr/src/uts/common/gssapi/mechs/krb5/include/etypes.h
usr/src/uts/common/gssapi/mechs/krb5/include/raw.h
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_auth.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_adata.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_addr.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_auth.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_cksum.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_ctx.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_key.c
usr/src/uts/common/gssapi/mechs/krb5/krb5/os/toffset.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_seed.c
usr/src/uts/common/gssapi/mechs/krb5/mech/util_seqnum.c
usr/src/uts/common/gssapi/mechs/krb5/mech/val_cred.c
usr/src/uts/common/io/cxgbe/*
usr/src/uts/common/io/iwn/THIRDPARTYLICENSE
usr/src/uts/common/io/iwn/THIRDPARTYLICENSE.descrip
usr/src/uts/common/io/iwn/fw-iw/THIRDPARTYLICENSE
usr/src/uts/common/io/iwn/fw-iw/THIRDPARTYLICENSE.descrip
usr/src/uts/common/io/iwn/fw-iw/*.ucode
usr/src/uts/common/io/ixgbe/core/ixgbe_82598.c
usr/src/uts/common/io/ixgbe/core/ixgbe_82598.h
usr/src/uts/common/io/ixgbe/core/ixgbe_82599.c
usr/src/uts/common/io/ixgbe/core/ixgbe_82599.h
usr/src/uts/common/io/ixgbe/core/ixgbe_api.c
usr/src/uts/common/io/ixgbe/core/ixgbe_api.h
usr/src/uts/common/io/ixgbe/core/ixgbe_common.c
usr/src/uts/common/io/ixgbe/core/ixgbe_common.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb.c
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82598.c
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82598.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82599.c
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82599.h
usr/src/uts/common/io/ixgbe/core/ixgbe_mbx.c
usr/src/uts/common/io/ixgbe/core/ixgbe_mbx.h
usr/src/uts/common/io/ixgbe/core/ixgbe_osdep.h
usr/src/uts/common/io/ixgbe/core/ixgbe_phy.c
usr/src/uts/common/io/ixgbe/core/ixgbe_phy.h
usr/src/uts/common/io/ixgbe/core/ixgbe_type.h
usr/src/uts/common/io/ixgbe/core/ixgbe_vf.c
usr/src/uts/common/io/ixgbe/core/ixgbe_vf.h
```



```
usr/src/uts/common/io/ixgbe/core/ixgbe_x540.c
usr/src/uts/common/io/ixgbe/core/ixgbe_x540.h
usr/src/uts/common/io/ixgbe/core/ixgbe_x550.c
usr/src/uts/common/io/ixgbe/core/ixgbe_x550.h
usr/src/uts/common/fs/zfs/THIRDPARTYLICENSE.lz4.descrip
usr/src/uts/common/fs/zfs/THIRDPARTYLICENSE.cityhash.descrip
usr/src/uts/common/sys/THIRDPARTYLICENSE.firmload
usr/src/uts/common/sys/THIRDPARTYLICENSE.firmload.descrip
usr/src/uts/common/sys/scsi/adapters/mpt_sas/mpi/*
usr/src/uts/sparc/nsmb/ioc_check.ref
```

```
************************************************************
   61629 Fri Dec 21 14:59:53 2018
new/exception_lists/cstyle
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
************************************************************
   1 usr/src/boot/*
   2 usr/src/cmd/acpi/acpidump/acpidump.h
   3 usr/src/cmd/acpi/acpidump/apdump.c
   4 usr/src/cmd/acpi/acpidump/apfiles.c
   5 usr/src/cmd/acpi/acpidump/apmain.c
   6 usr/src/cmd/acpi/acpidump/osunixdir.c
   7 usr/src/cmd/acpi/acpidump/tbprint.c
   8 usr/src/cmd/acpi/acpidump/tbxfroot.c
   9 usr/src/cmd/acpi/acpidump/utbuffer.c
  10 usr/src/cmd/acpi/acpixtract/acpixtract.[ch]
  11 usr/src/cmd/acpi/acpixtract/axmain.[ch]
  12 usr/src/cmd/acpi/acpixtract/axutils.[ch]
  13 usr/src/cmd/acpi/common/getopt.c
  14 usr/src/cmd/acpi/common/utascii.c
  15 usr/src/cmd/acpi/common/utdebug.c
  16 usr/src/cmd/acpi/common/utexcep.c
  17 usr/src/cmd/acpi/common/utglobal.c
  18 usr/src/cmd/acpi/common/utmath.c
  19 usr/src/cmd/acpi/common/utnonansi.c
  20 usr/src/cmd/acpi/common/utprint.c
  21 usr/src/cmd/acpi/common/utxferror.c
  22 usr/src/cmd/cmd-inet/usr.bin/dns-sd/ClientCommon.[ch]
  23 usr/src/cmd/cmd-inet/usr.bin/dns-sd/dns-sd.c
  24 usr/src/cmd/cmd-inet/usr.lib/mdnsd/CryptoAlg.[ch]
  25 usr/src/cmd/cmd-inet/usr.lib/mdnsd/DNSCommon.[ch]
  26 usr/src/cmd/cmd-inet/usr.lib/mdnsd/DNSDigest.[ch]
  27 usr/src/cmd/cmd-inet/usr.lib/mdnsd/PosixDaemon.c
  28 usr/src/cmd/cmd-inet/usr.lib/mdnsd/PlatformCommon.[ch]
  29 usr/src/cmd/cmd-inet/usr.lib/mdnsd/GenLinkedList.[ch]
  30 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNS.[ch]
  31 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSDebug.[ch]
  32 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSEmbeddedAPI.h
  33 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSPosix.[ch]
  34 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSUNP.[ch]
  35 usr/src/cmd/cmd-inet/usr.lib/mdnsd/nsec.h
  36 usr/src/cmd/cmd-inet/usr.lib/mdnsd/anonymous.[ch]
  37 usr/src/cmd/cmd-inet/usr.lib/mdnsd/dnssec.h
  38 usr/src/cmd/cmd-inet/usr.lib/mdnsd/uDNS.[ch]
  39 usr/src/cmd/cmd-inet/usr.lib/mdnsd/uds_daemon.[ch]
  40 usr/src/cmd/hal/tools/hal_set_property.c
  41 usr/src/cmd/krb5/kadmin/cli/kadmin_ct.c
  42 usr/src/cmd/krb5/kadmin/cli/kadmin.c
  43 usr/src/cmd/krb5/kadmin/cli/kadmin.h
  44 usr/src/cmd/krb5/kadmin/cli/keytab.c
  45 usr/src/cmd/krb5/kadmin/cli/ss_wrapper.c
  46 usr/src/cmd/krb5/kadmin/dbutil/dump.c
  47 usr/src/cmd/krb5/kadmin/dbutil/import_err.h
  48 usr/src/cmd/krb5/kadmin/dbutil/kadm5_create.c
  49 usr/src/cmd/krb5/kadmin/dbutil/kdb5_create.c
  50 usr/src/cmd/krb5/kadmin/dbutil/kdb5_destroy.c
  51 usr/src/cmd/krb5/kadmin/dbutil/kdb5_stash.c
  52 usr/src/cmd/krb5/kadmin/dbutil/kdb5_util.c
  53 usr/src/cmd/krb5/kadmin/dbutil/kdb5_util.h
  54 usr/src/cmd/krb5/kadmin/dbutil/nstrtok.h
  55 usr/src/cmd/krb5/kadmin/dbutil/ovload.c
  56 usr/src/cmd/krb5/kadmin/dbutil/string_table.c
  57 usr/src/cmd/krb5/kadmin/dbutil/string_table.h
  58 usr/src/cmd/krb5/kadmin/dbutil/strtok.c
  59 usr/src/cmd/krb5/kadmin/dbutil/util.c
  60 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd_strings.h
```



```
  61 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd.c
  62 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd.h
  63 usr/src/cmd/krb5/kadmin/kpasswd/tty_kpasswd.c
  64 usr/src/cmd/krb5/kadmin/ktutil/ktutil_ct.c
  65 usr/src/cmd/krb5/kadmin/ktutil/ktutil_funcs.c
  66 usr/src/cmd/krb5/kadmin/ktutil/ktutil.c
  67 usr/src/cmd/krb5/kadmin/ktutil/ktutil.h
  68 usr/src/cmd/krb5/kadmin/server/kadm_rpc_svc.c
  69 usr/src/cmd/krb5/kadmin/server/misc.c
  70 usr/src/cmd/krb5/kadmin/server/misc.h
  71 usr/src/cmd/krb5/kadmin/server/ovsec_kadmd.c
  72 usr/src/cmd/krb5/kadmin/server/server_glue_v1.c
  73 usr/src/cmd/krb5/kadmin/server/server_stubs.c
  74 usr/src/cmd/krb5/kdestroy/kdestroy.c
  75 usr/src/cmd/krb5/kinit/kinit.c
  76 usr/src/cmd/krb5/klist/klist.c
  77 usr/src/cmd/krb5/krb5kdc/dispatch.c
  78 usr/src/cmd/krb5/krb5kdc/do_as_req.c
  79 usr/src/cmd/krb5/krb5kdc/do_tgs_req.c
  80 usr/src/cmd/krb5/krb5kdc/extern.c
  81 usr/src/cmd/krb5/krb5kdc/extern.h
  82 usr/src/cmd/krb5/krb5kdc/kdc_preauth.c
  83 usr/src/cmd/krb5/krb5kdc/kdc_util.c
  84 usr/src/cmd/krb5/krb5kdc/kdc_util.h
  85 usr/src/cmd/krb5/krb5kdc/main.c
  86 usr/src/cmd/krb5/krb5kdc/network.c
  87 usr/src/cmd/krb5/krb5kdc/policy.c
  88 usr/src/cmd/krb5/krb5kdc/policy.h
  89 usr/src/cmd/krb5/krb5kdc/replay.c
  90 usr/src/cmd/krb5/krb5kdc/sock2p.c
  91 usr/src/cmd/krb5/ldap_util/kdb5_ldap_list.c
  92 usr/src/cmd/krb5/ldap_util/kdb5_ldap_list.h
  93 usr/src/cmd/krb5/ldap_util/kdb5_ldap_policy.c
  94 usr/src/cmd/krb5/ldap_util/kdb5_ldap_policy.h
  95 usr/src/cmd/krb5/ldap_util/kdb5_ldap_realm.c
  96 usr/src/cmd/krb5/ldap_util/kdb5_ldap_realm.h
  97 usr/src/cmd/krb5/ldap_util/kdb5_ldap_services.c
  98 usr/src/cmd/krb5/ldap_util/kdb5_ldap_services.h
  99 usr/src/cmd/krb5/ldap_util/kdb5_ldap_util.c
 100 usr/src/cmd/krb5/ldap_util/kdb5_ldap_util.h
 101 usr/src/cmd/krb5/slave/kprop.c
 102 usr/src/cmd/krb5/slave/kprop.h
 103 usr/src/cmd/krb5/slave/kpropd.c
 104 usr/src/cmd/mandoc
 105 usr/src/cmd/ttymon/sttyparse.c
 106 usr/src/common/ficl/ficltokens.h
 107 usr/src/common/bzip2/bzlib.h
 108 usr/src/common/bzip2/crctable.c
 109 usr/src/common/bzip2/randtable.c
 110 usr/src/common/bzip2/blocksort.c
 111 usr/src/common/bzip2/compress.c
 112 usr/src/common/bzip2/bzlib.c
 113 usr/src/common/bzip2/decompress.c
 114 usr/src/common/bzip2/bzlib_private.h
 115 usr/src/common/bzip2/huffman.c
 116 usr/src/common/crypto/chacha/chacha.c
 117 usr/src/grub/grub-0.97/grub/asmstub.c
 118 usr/src/grub/grub-0.97/stage2/bios.c
 119 usr/src/grub/grub-0.97/stage2/builtins.c
 120 usr/src/grub/grub-0.97/stage2/char_io.c
 121 usr/src/grub/grub-0.97/stage2/disk_io.c
 122 usr/src/grub/grub-0.97/stage2/gpt.h
 123 usr/src/grub/grub-0.97/stage2/moddiv.c
 124 usr/src/grub/grub-0.97/stage2/pc_slice.h
 125 usr/src/grub/grub-0.97/stage2/shared.h
 126 usr/src/grub/grub-0.97/stage2/stage1_5.c
```

```
usr/src/lib/libc/port/gen/arc4random_uniform.c
usr/src/lib/gss_mechs/mech_krb5/crypto/aes/aes_s2k.c
usr/src/lib/gss_mechs/mech_krb5/crypto/cksumtype_to_string.c
usr/src/lib/gss_mechs/mech_krb5/crypto/coll_proof_cksum.c
usr/src/lib/gss_mechs/mech_krb5/crypto/crc32/crc.c
usr/src/lib/gss_mechs/mech_krb5/crypto/des/afsstring2key.c
usr/src/lib/gss_mechs/mech_krb5/crypto/des/string2key.c
usr/src/lib/gss_mechs/mech_krb5/crypto/dk/stringtokey.c
usr/src/lib/gss_mechs/mech_krb5/crypto/enctype_compare.c
usr/src/lib/gss_mechs/mech_krb5/crypto/enctype_to_string.c
usr/src/lib/gss_mechs/mech_krb5/crypto/hash_provider/hash_md5.c
usr/src/lib/gss_mechs/mech_krb5/crypto/hash_provider/hash_sha1.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keyed_checksum_types.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keyed_cksum.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keyhash_provider/hmac_md5.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keyhash_provider/k5_md5des.c
usr/src/lib/gss_mechs/mech_krb5/crypto/keylengths.c
usr/src/lib/gss_mechs/mech_krb5/crypto/make_random_key.c
usr/src/lib/gss_mechs/mech_krb5/crypto/md4/md4.c
usr/src/lib/gss_mechs/mech_krb5/crypto/old_api_glue.c
usr/src/lib/gss_mechs/mech_krb5/crypto/old/des_stringtokey.c
usr/src/lib/gss_mechs/mech_krb5/crypto/pbkdf2.c
usr/src/lib/gss_mechs/mech_krb5/crypto/random_to_key.c
usr/src/lib/gss_mechs/mech_krb5/crypto/state.c
usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_cksumtype.c
usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_enctype.c
usr/src/lib/gss_mechs/mech_krb5/crypto/string_to_key.c
usr/src/lib/gss_mechs/mech_krb5/crypto/valid_cksumtype.c
usr/src/lib/gss_mechs/mech_krb5/crypto/valid_enctype.c
usr/src/lib/gss_mechs/mech_krb5/et/com_err.c
usr/src/lib/gss_mechs/mech_krb5/et/error_message.c
usr/src/lib/gss_mechs/mech_krb5/et/error_table.h
usr/src/lib/gss_mechs/mech_krb5/et/internal.h
usr/src/lib/gss_mechs/mech_krb5/et/mit-sipb-copyright.h
usr/src/lib/gss_mechs/mech_krb5/include/cache-addrinfo.h
usr/src/lib/gss_mechs/mech_krb5/include/cm.h
usr/src/lib/gss_mechs/mech_krb5/include/com_err.h
usr/src/lib/gss_mechs/mech_krb5/include/db-config.h
usr/src/lib/gss_mechs/mech_krb5/include/db.h
usr/src/lib/gss_mechs/mech_krb5/include/fake-addrinfo.h
usr/src/lib/gss_mechs/mech_krb5/include/foreachaddr.h
usr/src/lib/gss_mechs/mech_krb5/include/k5-int-pkinit.h
usr/src/lib/gss_mechs/mech_krb5/include/k5-utf8.h
usr/src/lib/gss_mechs/mech_krb5/include/kdb_kt.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5_libinit.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_defs.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_proto.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/copyright.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-err.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-plugin.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/kdb_dbc.h
usr/src/lib/gss_mechs/mech_krb5/include/krb5/kdb.h
usr/src/lib/gss_mechs/mech_krb5/include/locate_plugin.h
usr/src/lib/gss_mechs/mech_krb5/include/osconf.h
usr/src/lib/gss_mechs/mech_krb5/include/port-sockets.h
usr/src/lib/gss_mechs/mech_krb5/include/preauth_plugin.h
usr/src/lib/gss_mechs/mech_krb5/include/socket-utils.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.h
```



```
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1buf.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_decode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krb5_encode.c
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krbasn1.h
usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/ldap_key_seq.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_memory.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc_retr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccbase.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cccopy.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccdefops.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ccfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/fcc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/scc.h
usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/ser_cc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/error_tables/adm_err.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/file/ktfile.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt_srvtab.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktadd.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktbase.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktfr_entry.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/ktremove.c
usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/read_servi.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_comp.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_order.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/addr_srch.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/appdefault.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_pr_ext.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/bld_princ.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/chk_trans.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cleanup.h
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/conv_princ.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_addrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_creds.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_data.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/copy_tick.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cp_key_cnt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decode_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/decrypt_tk.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/deltat.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/enc_helper.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encode_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/encrypt_tk.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/free_rtree.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/fwd_tgt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gc_frm_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gc_via_tkt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gen_seqnum.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gen_subkey.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/get_creds.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/get_in_tkt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_keytab.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_opt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/gic_pwd.c
```

```
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/init_keyblock.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/int-proto.h
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kdc_rep_dc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/kerrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_error.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_priv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_rep.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_req_ext.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_req.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/mk_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/pac.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/pr_to_salt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/preauth.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/preauth2.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/princ_comp.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_cred.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_error.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_priv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_rep.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_req_dec.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_req.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/rd_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/recvauth.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/send_tgs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/sendauth.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/set_realm.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/srv_rcache.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/str_conv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/tgtname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/valid_times.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/vic_opt.c
usr/src/lib/gss_mechs/mech_krb5/krb5/krb/walk_rtree.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/accessor.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/an_to_ln.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/ccdefname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/changepw.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/def_realm.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnsglue.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnsglue.h
usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnssrv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/foreachaddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_hstrl.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/free_krbhs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/full_ipadr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_port.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gen_rname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/genaddrs.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/get_krbhst.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/gmt_mktime.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/hostaddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/hst_realm.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/ktdefname.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/kuserok.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/localaddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/locate_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/lock_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/mk_faddr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_read.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/net_write.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/os-proto.h
usr/src/lib/gss_mechs/mech_krb5/krb5/os/osconfig.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/port2ip.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/prompter.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/promptusr.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_msg.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/read_pwd.c
```



```
usr/src/lib/gss_mechs/mech_krb5/krb5/os/realm_dom.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/realm_iter.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/sendto_kdc.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/sn2princ.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/thread_safe.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/unlck_file.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/ustime.c
usr/src/lib/gss_mechs/mech_krb5/krb5/os/write_msg.c
usr/src/lib/gss_mechs/mech_krb5/krb5/posix/daemon.c
usr/src/lib/gss_mechs/mech_krb5/krb5/posix/setenv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_base.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_conv.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_io.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_none.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc-int.h
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rcdef.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rcfns.c
usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/ser_rc.c
usr/src/lib/gss_mechs/mech_krb5/mech/accept_sec_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/acquire_cred_with_pw.c
usr/src/lib/gss_mechs/mech_krb5/mech/acquire_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/add_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/compare_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/context_time.c
usr/src/lib/gss_mechs/mech_krb5/mech/copy_ccache.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_com_err_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_major_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/disp_status.c
usr/src/lib/gss_mechs/mech_krb5/mech/export_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/export_sec_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/get_tkt_flags.c
usr/src/lib/gss_mechs/mech_krb5/mech/gss_libinit.h
usr/src/lib/gss_mechs/mech_krb5/mech/import_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/indicate_mechs.c
usr/src/lib/gss_mechs/mech_krb5/mech/init_sec_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/inq_names.c
usr/src/lib/gss_mechs/mech_krb5/mech/krb5_gss_glue.c
usr/src/lib/gss_mechs/mech_krb5/mech/lucid_context.c
usr/src/lib/gss_mechs/mech_krb5/mech/oid_ops.c
usr/src/lib/gss_mechs/mech_krb5/mech/process_context_token.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_buffer.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_cred.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_name.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_oid_set.c
usr/src/lib/gss_mechs/mech_krb5/mech/rel_oid.c
usr/src/lib/gss_mechs/mech_krb5/mech/set_allowable_enctypes.c
usr/src/lib/gss_mechs/mech_krb5/mech/set_ccache.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_buffer_set.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_buffer.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_cksum.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_ctxsetup.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_dup.c
usr/src/lib/gss_mechs/mech_krb5/mech/util_localhost.c
usr/src/lib/gss_mechs/mech_krb5/mech/utl_nohash_validate.c
usr/src/lib/gss_mechs/mech_krb5/profile/prof_err.h
usr/src/lib/gss_mechs/mech_krb5/profile/prof_get.c
usr/src/lib/gss_mechs/mech_krb5/profile/prof_set.c
usr/src/lib/gss_mechs/mech_krb5/support/errors.c
usr/src/lib/gss_mechs/mech_krb5/support/fake-addrinfo.c
usr/src/lib/gss_mechs/mech_krb5/support/init-addrinfo.c
usr/src/lib/gss_mechs/mech_krb5/support/plugins.c
usr/src/lib/gss_mechs/mech_krb5/support/supp-int.h
usr/src/lib/gss_mechs/mech_krb5/support/threads.c
```

usr/src/lib/gss_mechs/mech_krb5/support/utf8_conv.c
usr/src/lib/gss_mechs/mech_krb5/support/utf8.c
usr/src/lib/krb5/dyn/dyn_append.c
usr/src/lib/krb5/dyn/dyn_create.c
usr/src/lib/krb5/dyn/dyn_debug.c
usr/src/lib/krb5/dyn/dyn_delete.c
usr/src/lib/krb5/dyn/dyn_initzero.c
usr/src/lib/krb5/dyn/dyn_insert.c
usr/src/lib/krb5/dyn/dyn_paranoid.c
usr/src/lib/krb5/dyn/dyn_put.c
usr/src/lib/krb5/dyn/dyn_realloc.c
usr/src/lib/krb5/dyn/dyn_size.c
usr/src/lib/krb5/kadm5/admin_internal.h
usr/src/lib/krb5/kadm5/admin_xdr.h
usr/src/lib/krb5/kadm5/admin.h
usr/src/lib/krb5/kadm5/alt_prof.c
usr/src/lib/krb5/kadm5/chpass_util_strings.h
usr/src/lib/krb5/kadm5/chpass_util.c
usr/src/lib/krb5/kadm5/clnt/changepw.c
usr/src/lib/krb5/kadm5/clnt/client_handle.c
usr/src/lib/krb5/kadm5/clnt/client_init.c
usr/src/lib/krb5/kadm5/clnt/client_internal.h
usr/src/lib/krb5/kadm5/clnt/client_principal.c
usr/src/lib/krb5/kadm5/clnt/client_rpc.c
usr/src/lib/krb5/kadm5/clnt/clnt_chpass_util.c
usr/src/lib/krb5/kadm5/clnt/clnt_policy.c
usr/src/lib/krb5/kadm5/clnt/clnt_privs.c
usr/src/lib/krb5/kadm5/clnt/logger.c
usr/src/lib/krb5/kadm5/kadm_err.h
usr/src/lib/krb5/kadm5/kadm_rpc_xdr.c
usr/src/lib/krb5/kadm5/kadm_rpc.h
usr/src/lib/krb5/kadm5/misc_free.c
usr/src/lib/krb5/kadm5/server_internal.h
usr/src/lib/krb5/kadm5/srv/adb_xdr.c
usr/src/lib/krb5/kadm5/srv/chgpwd.c
usr/src/lib/krb5/kadm5/srv/logger.c
usr/src/lib/krb5/kadm5/srv/server_acl.c
usr/src/lib/krb5/kadm5/srv/server_acl.h
usr/src/lib/krb5/kadm5/srv/server_dict.c
usr/src/lib/krb5/kadm5/srv/server_handle.c
usr/src/lib/krb5/kadm5/srv/server_init.c
usr/src/lib/krb5/kadm5/srv/server_kdb.c
usr/src/lib/krb5/kadm5/srv/server_misc.c
usr/src/lib/krb5/kadm5/srv/svr_chpass_util.c
usr/src/lib/krb5/kadm5/srv/svr_iters.c
usr/src/lib/krb5/kadm5/srv/svr_misc_free.c
usr/src/lib/krb5/kadm5/srv/svr_policy.c
usr/src/lib/krb5/kadm5/srv/svr_principal.c
usr/src/lib/krb5/kadm5/srv/xdr_alloc.c
usr/src/lib/krb5/kadm5/str_conv.c
usr/src/lib/krb5/kdb/adb_err.h
usr/src/lib/krb5/kdb/decrypt_key.c
usr/src/lib/krb5/kdb/encrypt_key.c
usr/src/lib/krb5/kdb/kdb_cpw.c
usr/src/lib/krb5/kdb/kdb_default.c
usr/src/lib/krb5/kdb/kdb5.c
usr/src/lib/krb5/kdb/kdb5.h
usr/src/lib/krb5/kdb/keytab.c
usr/src/lib/krb5/plugins/kdb/db2/adb_openclose.c
usr/src/lib/krb5/plugins/kdb/db2/adb_policy.c
usr/src/lib/krb5/plugins/kdb/db2/db2_exp.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_compat.h
usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.h
usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.c
usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.h



usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_close.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_conv.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_debug.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_delete.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_get.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_open.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_overflow.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_page.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_put.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_search.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_seq.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_split.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/bt_utils.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/btree.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/extern.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/db/db.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/dbm.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/extern.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_bigkey.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_func.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_log2.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash_page.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hsearch.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/page.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/search.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-int.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-ndbm.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-queue.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/extern.h
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_close.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_delete.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_get.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_open.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_put.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_search.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_seq.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/rec_utils.c
usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/recno.h
usr/src/lib/krb5/plugins/kdb/db2/pol_xdr.c
usr/src/lib/krb5/plugins/kdb/db2/policy_db.h
usr/src/lib/krb5/plugins/kdb/ldap/ldap_exp.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap_conn.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_create.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_fetch_mkey.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_main.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_misc.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_misc.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal2.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.h

```
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_rights.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.c
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_clnt.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto_openssl.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto_openssl.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_identity.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_lib.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_matching.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_profile.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_srv.c
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit.h
usr/src/lib/krb5/ss/copyright.h
usr/src/lib/krb5/ss/data.c
usr/src/lib/krb5/ss/error.c
usr/src/lib/krb5/ss/execute_cmd.c
usr/src/lib/krb5/ss/help.c
usr/src/lib/krb5/ss/invocation.c
usr/src/lib/krb5/ss/list_rqs.c
usr/src/lib/krb5/ss/listen.c
usr/src/lib/krb5/ss/mit-sipb-copyright.h
usr/src/lib/krb5/ss/mk_cmds.c
usr/src/lib/krb5/ss/options.c
usr/src/lib/krb5/ss/pager.c
usr/src/lib/krb5/ss/parse.c
usr/src/lib/krb5/ss/prompt.c
usr/src/lib/krb5/ss/request_tbl.c
usr/src/lib/krb5/ss/requests.c
usr/src/lib/krb5/ss/ss_internal.h
usr/src/lib/krb5/ss/ss.h
usr/src/lib/krb5/ss/std_rqs.c
usr/src/lib/krb5/ss/utils.c
usr/src/lib/libdns_sd/common/dnssd_clientlib.c
usr/src/lib/libdns_sd/common/dnssd_clientstub.c
usr/src/lib/libdns_sd/common/dnssd_ipc.c
usr/src/lib/libdns_sd/common/dnssd_ipc.h
usr/src/lib/libdns_sd/common/dns_sd.h
usr/src/lib/libdns_sd/java/common/JNISupport.c
usr/src/lib/libgss/g_glue.c
usr/src/lib/libresolv2/common
usr/src/lib/librstp/common/base.h
usr/src/lib/librstp/common/choose.h
usr/src/lib/librstp/common/edge.c
usr/src/lib/librstp/common/edge.h
usr/src/lib/librstp/common/migrate.c
usr/src/lib/librstp/common/migrate.h
usr/src/lib/librstp/common/p2p.c
usr/src/lib/librstp/common/p2p.h
usr/src/lib/librstp/common/pcost.c
usr/src/lib/librstp/common/pcost.h
usr/src/lib/librstp/common/port.c
usr/src/lib/librstp/common/port.h
usr/src/lib/librstp/common/portinfo.c
usr/src/lib/librstp/common/portinfo.h
usr/src/lib/librstp/common/rolesel.c
usr/src/lib/librstp/common/rolesel.h
usr/src/lib/librstp/common/roletrns.c
```



```
usr/src/lib/librstp/common/roletrns.h
usr/src/lib/librstp/common/statmch.c
usr/src/lib/librstp/common/statmch.h
usr/src/lib/librstp/common/stp_bpdu.h
usr/src/lib/librstp/common/stp_in.c
usr/src/lib/librstp/common/stp_in.h
usr/src/lib/librstp/common/stp_to.h
usr/src/lib/librstp/common/stp_vectors.h
usr/src/lib/librstp/common/stpm.c
usr/src/lib/librstp/common/stpm.h
usr/src/lib/librstp/common/stpmgmt.c
usr/src/lib/librstp/common/sttrans.c
usr/src/lib/librstp/common/sttrans.h
usr/src/lib/librstp/common/times.c
usr/src/lib/librstp/common/times.h
usr/src/lib/librstp/common/topoch.c
usr/src/lib/librstp/common/topoch.h
usr/src/lib/librstp/common/transmit.c
usr/src/lib/librstp/common/transmit.h
usr/src/lib/librstp/common/uid_stp.h
usr/src/lib/librstp/common/vector.c
usr/src/lib/librstp/common/vector.h
usr/src/lib/libsmbfs/smb/derparse.c
usr/src/lib/libsmbfs/smb/derparse.h
usr/src/lib/libsmbfs/smb/spnego.c
usr/src/lib/libsmbfs/smb/spnegoparse.c
usr/src/lib/libsmbfs/smb/spnegoparse.h
usr/src/test/libc-tests/tests/regex/testregex.c
usr/src/test/os-tests/tests/sockfs/conn.c
usr/src/test/os-tests/tests/sockfs/dgram.c
usr/src/test/os-tests/tests/sockfs/drop_priv.c
usr/src/test/os-tests/tests/sockfs/sockpair.c
usr/src/tools/btxld/btx.h
usr/src/tools/btxld/btxld.c
usr/src/tools/btxld/elfh.c
usr/src/tools/btxld/elfh.h
usr/src/tools/btxld/imgact_aout.h
usr/src/tools/smatch/src/*
usr/src/uts/common/fs/zfs/lua/*
usr/src/uts/common/gssapi/gssapi.h
usr/src/uts/common/gssapi/mechs/krb5/crypto/block_size.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/checksum_length.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/cksumtypes.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/combine_keys.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/crc32/crc32.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/decrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/default_state.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/d3_cbc.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/f_cbc.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/f_parity.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/des/weak_key.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/checksum.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/derive.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/dk_decrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/dk/dk_encrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/arcfour_provider.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/des.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/enc_provider/des3.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/encrypt_length.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/encrypt.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/etypes.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_crc32.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_kmd5.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hash_provider/hash_ksha1.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/hmac.c
usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/descbc.c
```

```
655 usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/k_hmac_md5.c
656 usr/src/uts/common/gssapi/mechs/krb5/crypto/keyhash_provider/k5_kmd5des.c
657 usr/src/uts/common/gssapi/mechs/krb5/crypto/make_checksum.c
658 usr/src/uts/common/gssapi/mechs/krb5/crypto/mandatory_sumtype.c
659 usr/src/uts/common/gssapi/mechs/krb5/crypto/nfold.c
660 usr/src/uts/common/gssapi/mechs/krb5/crypto/old/old_decrypt.c
661 usr/src/uts/common/gssapi/mechs/krb5/crypto/old/old_encrypt.c
662 usr/src/uts/common/gssapi/mechs/krb5/crypto/prng.c
663 usr/src/uts/common/gssapi/mechs/krb5/crypto/raw/raw_decrypt.c
664 usr/src/uts/common/gssapi/mechs/krb5/crypto/raw/raw_encrypt.c
665 usr/src/uts/common/gssapi/mechs/krb5/crypto/verify_checksum.c
666 usr/src/uts/common/gssapi/mechs/krb5/include/aes_s2k.h
667 usr/src/uts/common/gssapi/mechs/krb5/include/auth_con.h
668 usr/src/uts/common/gssapi/mechs/krb5/include/cksumtypes.h
669 usr/src/uts/common/gssapi/mechs/krb5/include/crc-32.h
670 usr/src/uts/common/gssapi/mechs/krb5/include/des_int.h
671 usr/src/uts/common/gssapi/mechs/krb5/include/dk.h
672 usr/src/uts/common/gssapi/mechs/krb5/include/enc_provider.h
673 usr/src/uts/common/gssapi/mechs/krb5/include/etypes.h
674 usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_generic.h
675 usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_krb5.h
676 usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_generic.h
677 usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_krb5.h
678 usr/src/uts/common/gssapi/mechs/krb5/include/hash_provider.h
679 usr/src/uts/common/gssapi/mechs/krb5/include/k5-int.h
680 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_16.h
681 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_32.h
682 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_64.h
683 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_16.h
684 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_32.h
685 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_64.h
686 usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform.h
687 usr/src/uts/common/gssapi/mechs/krb5/include/k5-thread.h
688 usr/src/uts/common/gssapi/mechs/krb5/include/keyhash_provider.h
689 usr/src/uts/common/gssapi/mechs/krb5/include/krb5.h
690 usr/src/uts/common/gssapi/mechs/krb5/include/old.h
691 usr/src/uts/common/gssapi/mechs/krb5/include/raw.h
692 usr/src/uts/common/gssapi/mechs/krb5/include/rsa-md4.h
693 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_athctr.c
694 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_auth.c
695 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_cksum.c
696 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_key.c
697 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/copy_princ.c
698 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/init_ctx.c
699 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/kfree.c
700 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/parse.c
701 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_actx.c
702 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_adata.c
703 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_addr.c
704 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_auth.c
705 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_cksum.c
706 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_ctx.c
707 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_key.c
708 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/ser_princ.c
709 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/serialize.c
710 usr/src/uts/common/gssapi/mechs/krb5/krb5/krb/unparse.c
711 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/c_ustime.c
712 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/init_os_ctx.c
713 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/timeofday.c
714 usr/src/uts/common/gssapi/mechs/krb5/krb5/os/toffset.c
715 usr/src/uts/common/gssapi/mechs/krb5/mech/delete_sec_context.c
716 usr/src/uts/common/gssapi/mechs/krb5/mech/gssapi_krb5.c
717 usr/src/uts/common/gssapi/mechs/krb5/mech/import_sec_context.c
718 usr/src/uts/common/gssapi/mechs/krb5/mech/k5seal.c
719 usr/src/uts/common/gssapi/mechs/krb5/mech/k5sealv3.c
720 usr/src/uts/common/gssapi/mechs/krb5/mech/k5unseal.c
```



```
721 usr/src/uts/common/gssapi/mechs/krb5/mech/seal.c
722 usr/src/uts/common/gssapi/mechs/krb5/mech/ser_sctx.c
723 usr/src/uts/common/gssapi/mechs/krb5/mech/sign.c
724 usr/src/uts/common/gssapi/mechs/krb5/mech/unseal.c
725 usr/src/uts/common/gssapi/mechs/krb5/mech/util_crypt.c
726 usr/src/uts/common/gssapi/mechs/krb5/mech/util_ordering.c
727 usr/src/uts/common/gssapi/mechs/krb5/mech/util_seed.c
728 usr/src/uts/common/gssapi/mechs/krb5/mech/util_seqnum.c
729 usr/src/uts/common/gssapi/mechs/krb5/mech/util_set.c
730 usr/src/uts/common/gssapi/mechs/krb5/mech/util_token.c
731 usr/src/uts/common/gssapi/mechs/krb5/mech/util_validate.c
732 usr/src/uts/common/gssapi/mechs/krb5/mech/val_cred.c
733 usr/src/uts/common/gssapi/mechs/krb5/mech/verify.c
734 usr/src/uts/common/gssapi/mechs/krb5/mech/wrap_size_limit.c
735 usr/src/uts/common/io/bnxe/577xx/common/bnxe_clc.c
736 usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_common.h
737 usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_init_ops.h
738 usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_init.h
739 usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_sp_verbs.c
740 usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_sp_verbs.h
741 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/bcm_utils.h
742 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/cyclic_oper.h
743 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/debug.h
744 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/l4/mm_l4if.h
745 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/l5/mm_l5if.h
746 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_dos.h
747 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_linux.h
748 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_ndismono.h
749 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_solaris.h
750 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_uefi.h
751 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_user_mode_debug.h
752 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_vbd.h
753 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm.h
754 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/utils.h
755 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/vm/hw_channel.h
756 usr/src/uts/common/io/bnxe/577xx/drivers/common/include/vm/vfpf_if.h
757 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/bd_chain_st.h
758 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/bd_chain.h
759 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/bnxe_context.c
760 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/bnxe_hw_debug.c
761 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/context.h
762 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/hw_debug.h
763 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/hw_setup.c
764 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/l2_dbg.c
765 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_dcbx_mp.c
766 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_dcbx_mp.h
767 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_dcbx.c
768 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_desc.h
769 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_devinfo.c
770 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_dmae.c
771 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_dmae.h
772 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_er.c
773 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_hw_access.c
774 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_hw_attn.c
775 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_hw_init_reset.c
776 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_main.c
777 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_mcp.c
778 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_niv.c
779 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_nvram.c
780 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_pf.c
781 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_phy.c
782 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_power.c
783 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_recv.c
784 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_resc.c
785 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_sb.c
786 usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_send.c
```

usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_sp_req_mgr.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_sp_req_mgr.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_sp.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_stats.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_util.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/5710_hsi.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57710_init_values.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57710_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57711_init_values.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57711_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57712_init_values.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57712_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/577xx_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_fcoe_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_fcoe_rfc_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_iscsi_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_iscsi_rfc_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_l5cm_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/bnxe_fw_funcs.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/eth_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/everest_iscsi_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/everest_l5cm_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/fcoe_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/fw_defs.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/init_defs.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/microcode_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/rdma_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/tcp_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/toe_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/command.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/common_uif.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm_stats.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm5710_hsi.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm5710.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/include/lm_l4if.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/include/lm_l4st.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4fp.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4fp.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4rx.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4sp.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4sp.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4tx.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l5/include/lm_l5if.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l5/include/lm_l5st.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l5/lm_l5.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l5/lm_l5sp.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/basic_vf/lm_vf.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/basic_vf/lm_vf.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/channel_vf/lm_vf.c
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/channel_vf/lm_vf.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/common/lm_vf_common.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/57712_reg.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/aeu_inputs.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/bigmac_addresses.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/cdu_def.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/clc_reg.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/clc.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/dbg_bus.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/dbu_reg_Driver.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/dmae_clients.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/emac_reg_driver.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/general_atten_bits.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/grc_addr.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/hw_dump.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/igu_def.h



usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/misc_bits.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/pcics_reg_driver.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/phy_reg.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/prs_flags.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/append.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/bdn.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/code_swap_def.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/dev_info.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mac_drv_info.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mac_stats.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mac_stx.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mcp_fio.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mcp_shmem.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/multi_thread_def.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/ncsi_basic_types.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/ncsi_cmds.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/nvm_map.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/shmem.h
usr/src/uts/common/io/bnxe/577xx/include/bcmtype.h
usr/src/uts/common/io/bnxe/577xx/include/iscsi_info.h
usr/src/uts/common/io/bnxe/577xx/include/l4debug.h
usr/src/uts/common/io/bnxe/577xx/include/l4states.h
usr/src/uts/common/io/bnxe/577xx/include/license.h
usr/src/uts/common/io/bnxe/577xx/include/listq.h
usr/src/uts/common/io/bnxe/577xx/include/lm_defs.h
usr/src/uts/common/io/bnxe/bnxe_binding.h
usr/src/uts/common/io/bnxe/bnxe_cfg.c
usr/src/uts/common/io/bnxe/bnxe_debug.c
usr/src/uts/common/io/bnxe/bnxe_debug.h
usr/src/uts/common/io/bnxe/bnxe_fcoe.c
usr/src/uts/common/io/bnxe/bnxe_gld.c
usr/src/uts/common/io/bnxe/bnxe_hw.c
usr/src/uts/common/io/bnxe/bnxe_intr.c
usr/src/uts/common/io/bnxe/bnxe_kstat.c
usr/src/uts/common/io/bnxe/bnxe_lint.c
usr/src/uts/common/io/bnxe/bnxe_lock.c
usr/src/uts/common/io/bnxe/bnxe_main.c
usr/src/uts/common/io/bnxe/bnxe_mm_l4.c
usr/src/uts/common/io/bnxe/bnxe_mm_l5.c
usr/src/uts/common/io/bnxe/bnxe_mm.c
usr/src/uts/common/io/bnxe/bnxe_rr.c
usr/src/uts/common/io/bnxe/bnxe_rx.c
usr/src/uts/common/io/bnxe/bnxe_timer.c
usr/src/uts/common/io/bnxe/bnxe_tx.c
usr/src/uts/common/io/bnxe/bnxe_workq.c
usr/src/uts/common/io/bnxe/bnxe.h
usr/src/uts/common/io/bnxe/version.h
usr/src/uts/common/io/bge/bge_main2.c
usr/src/uts/common/io/bge/bge_chip2.c
usr/src/uts/common/io/bge/bge_mii.c
usr/src/uts/common/io/bge/bge_kstats.c
usr/src/uts/common/io/bge/bge_impl.h
usr/src/uts/common/io/bge/bge_send.c
usr/src/uts/common/io/bge/bge_hw.h
usr/src/uts/common/io/cxgbe/*
usr/src/uts/common/io/e1000api/e1000_80003es2lan.c
usr/src/uts/common/io/e1000api/e1000_80003es2lan.h
usr/src/uts/common/io/e1000api/e1000_82540.c
usr/src/uts/common/io/e1000api/e1000_82541.c
usr/src/uts/common/io/e1000api/e1000_82541.h
usr/src/uts/common/io/e1000api/e1000_82542.c
usr/src/uts/common/io/e1000api/e1000_82543.c
usr/src/uts/common/io/e1000api/e1000_82543.h
usr/src/uts/common/io/e1000api/e1000_82571.c
usr/src/uts/common/io/e1000api/e1000_82571.h
usr/src/uts/common/io/e1000api/e1000_82575.c

usr/src/uts/common/io/e1000api/e1000_82575.h
usr/src/uts/common/io/e1000api/e1000_api.c
usr/src/uts/common/io/e1000api/e1000_api.h
usr/src/uts/common/io/e1000api/e1000_defines.h
usr/src/uts/common/io/e1000api/e1000_hw.h
usr/src/uts/common/io/e1000api/e1000_i210.c
usr/src/uts/common/io/e1000api/e1000_i210.h
usr/src/uts/common/io/e1000api/e1000_ich8lan.c
usr/src/uts/common/io/e1000api/e1000_ich8lan.h
usr/src/uts/common/io/e1000api/e1000_mac.c
usr/src/uts/common/io/e1000api/e1000_mac.h
usr/src/uts/common/io/e1000api/e1000_manage.c
usr/src/uts/common/io/e1000api/e1000_manage.h
usr/src/uts/common/io/e1000api/e1000_mbx.c
usr/src/uts/common/io/e1000api/e1000_mbx.h
usr/src/uts/common/io/e1000api/e1000_nvm.c
usr/src/uts/common/io/e1000api/e1000_nvm.h
usr/src/uts/common/io/e1000api/e1000_phy.c
usr/src/uts/common/io/e1000api/e1000_phy.h
usr/src/uts/common/io/e1000api/e1000_regs.h
usr/src/uts/common/io/e1000api/e1000_vf.c
usr/src/uts/common/io/e1000api/e1000_vf.h
usr/src/uts/common/io/i40e/core/i40e_adminq_cmd.h
usr/src/uts/common/io/i40e/core/i40e_adminq.c
usr/src/uts/common/io/i40e/core/i40e_adminq.h
usr/src/uts/common/io/i40e/core/i40e_alloc.h
usr/src/uts/common/io/i40e/core/i40e_common.c
usr/src/uts/common/io/i40e/core/i40e_devids.h
usr/src/uts/common/io/i40e/core/i40e_hmc.c
usr/src/uts/common/io/i40e/core/i40e_hmc.h
usr/src/uts/common/io/i40e/core/i40e_lan_hmc.c
usr/src/uts/common/io/i40e/core/i40e_lan_hmc.h
usr/src/uts/common/io/i40e/core/i40e_nvm.c
usr/src/uts/common/io/i40e/core/i40e_prototype.h
usr/src/uts/common/io/i40e/core/i40e_register.h
usr/src/uts/common/io/i40e/core/i40e_status.h
usr/src/uts/common/io/i40e/core/i40e_type.h
usr/src/uts/common/io/i40e/core/i40e_virtchnl.h
usr/src/uts/common/io/iwn/if_iwn.c
usr/src/uts/common/io/iwn/if_iwnreg.h
usr/src/uts/common/io/iwn/if_iwnvar.h
usr/src/uts/common/io/ixgbe/core/ixgbe_82598.c
usr/src/uts/common/io/ixgbe/core/ixgbe_82598.h
usr/src/uts/common/io/ixgbe/core/ixgbe_82599.c
usr/src/uts/common/io/ixgbe/core/ixgbe_82599.h
usr/src/uts/common/io/ixgbe/core/ixgbe_api.c
usr/src/uts/common/io/ixgbe/core/ixgbe_api.h
usr/src/uts/common/io/ixgbe/core/ixgbe_common.c
usr/src/uts/common/io/ixgbe/core/ixgbe_common.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb.c
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82598.c
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82598.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82599.c
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82599.h
usr/src/uts/common/io/ixgbe/core/ixgbe_mbx.c
usr/src/uts/common/io/ixgbe/core/ixgbe_mbx.h
usr/src/uts/common/io/ixgbe/core/ixgbe_osdep.h
usr/src/uts/common/io/ixgbe/core/ixgbe_phy.c
usr/src/uts/common/io/ixgbe/core/ixgbe_phy.h
usr/src/uts/common/io/ixgbe/core/ixgbe_type.h
usr/src/uts/common/io/ixgbe/core/ixgbe_vf.c
usr/src/uts/common/io/ixgbe/core/ixgbe_vf.h
usr/src/uts/common/io/ixgbe/core/ixgbe_x540.c
usr/src/uts/common/io/ixgbe/core/ixgbe_x540.h
usr/src/uts/common/io/ixgbe/core/ixgbe_x550.c



usr/src/uts/common/io/ixgbe/core/ixgbe_x550.h
usr/src/uts/common/io/qede/*
usr/src/uts/common/io/sfxge/common/ef10_ev.c
usr/src/uts/common/io/sfxge/common/ef10_filter.c
usr/src/uts/common/io/sfxge/common/ef10_impl.h
usr/src/uts/common/io/sfxge/common/ef10_intr.c
usr/src/uts/common/io/sfxge/common/ef10_mac.c
usr/src/uts/common/io/sfxge/common/ef10_mcdi.c
usr/src/uts/common/io/sfxge/common/ef10_nic.c
usr/src/uts/common/io/sfxge/common/ef10_nvram.c
usr/src/uts/common/io/sfxge/common/ef10_phy.c
usr/src/uts/common/io/sfxge/common/ef10_rx.c
usr/src/uts/common/io/sfxge/common/ef10_tlv_layout.h
usr/src/uts/common/io/sfxge/common/ef10_tx.c
usr/src/uts/common/io/sfxge/common/ef10_vpd.c
usr/src/uts/common/io/sfxge/common/efx.h
usr/src/uts/common/io/sfxge/common/efx_bootcfg.c
usr/src/uts/common/io/sfxge/common/efx_check.h
usr/src/uts/common/io/sfxge/common/efx_crc32.c
usr/src/uts/common/io/sfxge/common/efx_ev.c
usr/src/uts/common/io/sfxge/common/efx_filter.c
usr/src/uts/common/io/sfxge/common/efx_hash.c
usr/src/uts/common/io/sfxge/common/efx_impl.h
usr/src/uts/common/io/sfxge/common/efx_intr.c
usr/src/uts/common/io/sfxge/common/efx_lic.c
usr/src/uts/common/io/sfxge/common/efx_mac.c
usr/src/uts/common/io/sfxge/common/efx_mcdi.c
usr/src/uts/common/io/sfxge/common/efx_mcdi.h
usr/src/uts/common/io/sfxge/common/efx_mon.c
usr/src/uts/common/io/sfxge/common/efx_nic.c
usr/src/uts/common/io/sfxge/common/efx_nvram.c
usr/src/uts/common/io/sfxge/common/efx_phy.c
usr/src/uts/common/io/sfxge/common/efx_phy_ids.h
usr/src/uts/common/io/sfxge/common/efx_port.c
usr/src/uts/common/io/sfxge/common/efx_regs.h
usr/src/uts/common/io/sfxge/common/efx_regs_ef10.h
usr/src/uts/common/io/sfxge/common/efx_regs_mcdi.h
usr/src/uts/common/io/sfxge/common/efx_regs_pci.h
usr/src/uts/common/io/sfxge/common/efx_rx.c
usr/src/uts/common/io/sfxge/common/efx_sram.c
usr/src/uts/common/io/sfxge/common/efx_tx.c
usr/src/uts/common/io/sfxge/common/efx_types.h
usr/src/uts/common/io/sfxge/common/efx_vpd.c
usr/src/uts/common/io/sfxge/common/efx_wol.c
usr/src/uts/common/io/sfxge/common/hunt_impl.h
usr/src/uts/common/io/sfxge/common/hunt_nic.c
usr/src/uts/common/io/sfxge/common/hunt_phy.c
usr/src/uts/common/io/sfxge/common/mcdi_mon.c
usr/src/uts/common/io/sfxge/common/mcdi_mon.h
usr/src/uts/common/io/sfxge/common/medford_impl.h
usr/src/uts/common/io/sfxge/common/medford_nic.c
usr/src/uts/common/io/sfxge/common/siena_flash.h
usr/src/uts/common/io/sfxge/common/siena_impl.h
usr/src/uts/common/io/sfxge/common/siena_mac.c
usr/src/uts/common/io/sfxge/common/siena_mcdi.c
usr/src/uts/common/io/sfxge/common/siena_nic.c
usr/src/uts/common/io/sfxge/common/siena_nvram.c
usr/src/uts/common/io/sfxge/common/siena_phy.c
usr/src/uts/common/io/sfxge/common/siena_sram.c
usr/src/uts/common/io/sfxge/common/siena_vpd.c
usr/src/uts/common/sys/scsi/adapters/mpt_sas/mpi/*
usr/src/uts/intel/io/acpica/debugger/dbcmds.c
usr/src/uts/intel/io/acpica/debugger/dbdisply.c
usr/src/uts/intel/io/acpica/debugger/dbexec.c
usr/src/uts/intel/io/acpica/debugger/dbfileio.c
usr/src/uts/intel/io/acpica/debugger/dbhistry.c

```
1051 usr/src/uts/intel/io/acpica/debugger/dbinput.c
1052 usr/src/uts/intel/io/acpica/debugger/dbmethod.c
1053 usr/src/uts/intel/io/acpica/debugger/dbnames.c
1054 usr/src/uts/intel/io/acpica/debugger/dbstats.c
1055 usr/src/uts/intel/io/acpica/debugger/dbutils.c
1056 usr/src/uts/intel/io/acpica/debugger/dbxface.c
1057 usr/src/uts/intel/io/acpica/disassembler/dmbuffer.c
1058 usr/src/uts/intel/io/acpica/disassembler/dmnames.c
1059 usr/src/uts/intel/io/acpica/disassembler/dmobject.c
1060 usr/src/uts/intel/io/acpica/disassembler/dmopcode.c
1061 usr/src/uts/intel/io/acpica/disassembler/dmresrc.c
1062 usr/src/uts/intel/io/acpica/disassembler/dmresrcl.c
1063 usr/src/uts/intel/io/acpica/disassembler/dmresrcs.c
1064 usr/src/uts/intel/io/acpica/disassembler/dmutils.c
1065 usr/src/uts/intel/io/acpica/disassembler/dmwalk.c
1066 usr/src/uts/intel/io/acpica/dispatcher/dsargs.c
1067 usr/src/uts/intel/io/acpica/dispatcher/dscontrol.c
1068 usr/src/uts/intel/io/acpica/dispatcher/dsfield.c
1069 usr/src/uts/intel/io/acpica/dispatcher/dsinit.c
1070 usr/src/uts/intel/io/acpica/dispatcher/dsmethod.c
1071 usr/src/uts/intel/io/acpica/dispatcher/dsmthdat.c
1072 usr/src/uts/intel/io/acpica/dispatcher/dsobject.c
1073 usr/src/uts/intel/io/acpica/dispatcher/dsopcode.c
1074 usr/src/uts/intel/io/acpica/dispatcher/dsutils.c
1075 usr/src/uts/intel/io/acpica/dispatcher/dswexec.c
1076 usr/src/uts/intel/io/acpica/dispatcher/dswload.c
1077 usr/src/uts/intel/io/acpica/dispatcher/dswload2.c
1078 usr/src/uts/intel/io/acpica/dispatcher/dswscope.c
1079 usr/src/uts/intel/io/acpica/dispatcher/dswstate.c
1080 usr/src/uts/intel/io/acpica/events/evevent.c
1081 usr/src/uts/intel/io/acpica/events/evglock.c
1082 usr/src/uts/intel/io/acpica/events/evgpe.c
1083 usr/src/uts/intel/io/acpica/events/evgpeblk.c
1084 usr/src/uts/intel/io/acpica/events/evgpeinit.c
1085 usr/src/uts/intel/io/acpica/events/evgpeutil.c
1086 usr/src/uts/intel/io/acpica/events/evmisc.c
1087 usr/src/uts/intel/io/acpica/events/evregion.c
1088 usr/src/uts/intel/io/acpica/events/evrgnini.c
1089 usr/src/uts/intel/io/acpica/events/evsci.c
1090 usr/src/uts/intel/io/acpica/events/evxface.c
1091 usr/src/uts/intel/io/acpica/events/evxfevnt.c
1092 usr/src/uts/intel/io/acpica/events/evxfgpe.c
1093 usr/src/uts/intel/io/acpica/events/evxfregn.c
1094 usr/src/uts/intel/io/acpica/executer/exconfig.c
1095 usr/src/uts/intel/io/acpica/executer/exconvrt.c
1096 usr/src/uts/intel/io/acpica/executer/excreate.c
1097 usr/src/uts/intel/io/acpica/executer/exdebug.c
1098 usr/src/uts/intel/io/acpica/executer/exdump.c
1099 usr/src/uts/intel/io/acpica/executer/exfield.c
1100 usr/src/uts/intel/io/acpica/executer/exfldio.c
1101 usr/src/uts/intel/io/acpica/executer/exmisc.c
1102 usr/src/uts/intel/io/acpica/executer/exmutex.c
1103 usr/src/uts/intel/io/acpica/executer/exnames.c
1104 usr/src/uts/intel/io/acpica/executer/exoparg1.c
1105 usr/src/uts/intel/io/acpica/executer/exoparg2.c
1106 usr/src/uts/intel/io/acpica/executer/exoparg3.c
1107 usr/src/uts/intel/io/acpica/executer/exoparg6.c
1108 usr/src/uts/intel/io/acpica/executer/exprep.c
1109 usr/src/uts/intel/io/acpica/executer/exregion.c
1110 usr/src/uts/intel/io/acpica/executer/exresnte.c
1111 usr/src/uts/intel/io/acpica/executer/exresolv.c
1112 usr/src/uts/intel/io/acpica/executer/exresop.c
1113 usr/src/uts/intel/io/acpica/executer/exstore.c
1114 usr/src/uts/intel/io/acpica/executer/exstoren.c
1115 usr/src/uts/intel/io/acpica/executer/exstorob.c
1116 usr/src/uts/intel/io/acpica/executer/exsystem.c
```



```
1117 usr/src/uts/intel/io/acpica/executer/exutils.c
1118 usr/src/uts/intel/io/acpica/hardware/hwacpi.c
1119 usr/src/uts/intel/io/acpica/hardware/hwgpe.c
1120 usr/src/uts/intel/io/acpica/hardware/hwpci.c
1121 usr/src/uts/intel/io/acpica/hardware/hwregs.c
1122 usr/src/uts/intel/io/acpica/hardware/hwsleep.c
1123 usr/src/uts/intel/io/acpica/hardware/hwtimer.c
1124 usr/src/uts/intel/io/acpica/hardware/hwvalid.c
1125 usr/src/uts/intel/io/acpica/hardware/hwxface.c
1126 usr/src/uts/intel/io/acpica/namespace/nsaccess.c
1127 usr/src/uts/intel/io/acpica/namespace/nsalloc.c
1128 usr/src/uts/intel/io/acpica/namespace/nsdump.c
1129 usr/src/uts/intel/io/acpica/namespace/nsdumpdv.c
1130 usr/src/uts/intel/io/acpica/namespace/nseval.c
1131 usr/src/uts/intel/io/acpica/namespace/nsinit.c
1132 usr/src/uts/intel/io/acpica/namespace/nsload.c
1133 usr/src/uts/intel/io/acpica/namespace/nsnames.c
1134 usr/src/uts/intel/io/acpica/namespace/nsobject.c
1135 usr/src/uts/intel/io/acpica/namespace/nsparse.c
1136 usr/src/uts/intel/io/acpica/namespace/nspredef.c
1137 usr/src/uts/intel/io/acpica/namespace/nsrepair.c
1138 usr/src/uts/intel/io/acpica/namespace/nsrepair2.c
1139 usr/src/uts/intel/io/acpica/namespace/nssearch.c
1140 usr/src/uts/intel/io/acpica/namespace/nsutils.c
1141 usr/src/uts/intel/io/acpica/namespace/nswalk.c
1142 usr/src/uts/intel/io/acpica/namespace/nsxfeval.c
1143 usr/src/uts/intel/io/acpica/namespace/nsxfname.c
1144 usr/src/uts/intel/io/acpica/namespace/nsxfobj.c
1145 usr/src/uts/intel/io/acpica/parser/psargs.c
1146 usr/src/uts/intel/io/acpica/parser/psloop.c
1147 usr/src/uts/intel/io/acpica/parser/psopcode.c
1148 usr/src/uts/intel/io/acpica/parser/psparse.c
1149 usr/src/uts/intel/io/acpica/parser/psscope.c
1150 usr/src/uts/intel/io/acpica/parser/pstree.c
1151 usr/src/uts/intel/io/acpica/parser/psutils.c
1152 usr/src/uts/intel/io/acpica/parser/pswalk.c
1153 usr/src/uts/intel/io/acpica/parser/psxface.c
1154 usr/src/uts/intel/io/acpica/resources/rsaddr.c
1155 usr/src/uts/intel/io/acpica/resources/rscalc.c
1156 usr/src/uts/intel/io/acpica/resources/rscreate.c
1157 usr/src/uts/intel/io/acpica/resources/rsdump.c
1158 usr/src/uts/intel/io/acpica/resources/rsinfo.c
1159 usr/src/uts/intel/io/acpica/resources/rsio.c
1160 usr/src/uts/intel/io/acpica/resources/rsirq.c
1161 usr/src/uts/intel/io/acpica/resources/rslist.c
1162 usr/src/uts/intel/io/acpica/resources/rsmemory.c
1163 usr/src/uts/intel/io/acpica/resources/rsmisc.c
1164 usr/src/uts/intel/io/acpica/resources/rsutils.c
1165 usr/src/uts/intel/io/acpica/resources/rsxface.c
1166 usr/src/uts/intel/io/acpica/tables/tbfadt.c
1167 usr/src/uts/intel/io/acpica/tables/tbfind.c
1168 usr/src/uts/intel/io/acpica/tables/tbinstal.c
1169 usr/src/uts/intel/io/acpica/tables/tbutils.c
1170 usr/src/uts/intel/io/acpica/tables/tbxface.c
1171 usr/src/uts/intel/io/acpica/tables/tbxfroot.c
1172 usr/src/uts/intel/io/acpica/utilities/utalloc.c
1173 usr/src/uts/intel/io/acpica/utilities/utcache.c
1174 usr/src/uts/intel/io/acpica/utilities/utclib.c
1175 usr/src/uts/intel/io/acpica/utilities/utcopy.c
1176 usr/src/uts/intel/io/acpica/utilities/utdebug.c
1177 usr/src/uts/intel/io/acpica/utilities/utdecode.c
1178 usr/src/uts/intel/io/acpica/utilities/utdelete.c
1179 usr/src/uts/intel/io/acpica/utilities/uteval.c
1180 usr/src/uts/intel/io/acpica/utilities/utglobal.c
1181 usr/src/uts/intel/io/acpica/utilities/utids.c
1182 usr/src/uts/intel/io/acpica/utilities/utinit.c
```

```
1183 usr/src/uts/intel/io/acpica/utilities/utlock.c
1184 usr/src/uts/intel/io/acpica/utilities/utmath.c
1185 usr/src/uts/intel/io/acpica/utilities/utmisc.c
1186 usr/src/uts/intel/io/acpica/utilities/utmutex.c
1187 usr/src/uts/intel/io/acpica/utilities/utobject.c
1188 usr/src/uts/intel/io/acpica/utilities/utosi.c
1189 usr/src/uts/intel/io/acpica/utilities/utresrc.c
1190 usr/src/uts/intel/io/acpica/utilities/utstate.c
1191 usr/src/uts/intel/io/acpica/utilities/uttrack.c
1192 usr/src/uts/intel/io/acpica/utilities/utxface.c
1193 usr/src/uts/intel/io/acpica/utilities/utxferror.c
1194 usr/src/uts/intel/sys/acpi/acapps.h
1195 usr/src/uts/intel/sys/acpi/accommon.h
1196 usr/src/uts/intel/sys/acpi/acconfig.h
1197 usr/src/uts/intel/sys/acpi/acdebug.h
1198 usr/src/uts/intel/sys/acpi/acdisasm.h
1199 usr/src/uts/intel/sys/acpi/acdispat.h
1200 usr/src/uts/intel/sys/acpi/acevents.h
1201 usr/src/uts/intel/sys/acpi/acexcep.h
1202 usr/src/uts/intel/sys/acpi/acglobal.h
1203 usr/src/uts/intel/sys/acpi/achware.h
1204 usr/src/uts/intel/sys/acpi/acinterp.h
1205 usr/src/uts/intel/sys/acpi/aclocal.h
1206 usr/src/uts/intel/sys/acpi/acmacros.h
1207 usr/src/uts/intel/sys/acpi/acnames.h
1208 usr/src/uts/intel/sys/acpi/acnamesp.h
1209 usr/src/uts/intel/sys/acpi/acobject.h
1210 usr/src/uts/intel/sys/acpi/acopcode.h
1211 usr/src/uts/intel/sys/acpi/acoutput.h
1212 usr/src/uts/intel/sys/acpi/acparser.h
1213 usr/src/uts/intel/sys/acpi/acpi.h
1214 usr/src/uts/intel/sys/acpi/acpiosxf.h
1215 usr/src/uts/intel/sys/acpi/acpixf.h
1216 usr/src/uts/intel/sys/acpi/acpredef.h
1217 usr/src/uts/intel/sys/acpi/acresrc.h
1218 usr/src/uts/intel/sys/acpi/acrestyp.h
1219 usr/src/uts/intel/sys/acpi/acstruct.h
1220 usr/src/uts/intel/sys/acpi/actables.h
1221 usr/src/uts/intel/sys/acpi/actbl.h
1222 usr/src/uts/intel/sys/acpi/actbl1.h
1223 usr/src/uts/intel/sys/acpi/actbl2.h
1224 usr/src/uts/intel/sys/acpi/actypes.h
1225 usr/src/uts/intel/sys/acpi/acutils.h
1226 usr/src/uts/intel/sys/acpi/amlcode.h
1227 usr/src/uts/intel/sys/acpi/amlresrc.h
1228 usr/src/uts/intel/sys/acpi/platform/accygwin.h
1229 usr/src/uts/intel/sys/acpi/platform/acefi.h
1230 usr/src/uts/intel/sys/acpi/platform/acenv.h
1231 usr/src/uts/intel/sys/acpi/platform/acfreebsd.h
1232 usr/src/uts/intel/sys/acpi/platform/acgcc.h
1233 usr/src/uts/intel/sys/acpi/platform/acintel.h
1234 usr/src/uts/intel/sys/acpi/platform/aclinux.h
1235 usr/src/uts/intel/sys/acpi/platform/acmsvc.h
1236 usr/src/uts/intel/sys/acpi/platform/acnetbsd.h
1237 usr/src/uts/intel/sys/acpi/platform/acos2.h
1238 usr/src/uts/intel/sys/acpi/platform/acsolaris.h
1239 usr/src/uts/intel/sys/acpi/platform/acwin.h
1240 usr/src/uts/intel/sys/acpi/platform/acwin64.h
```

```
**********************************************************
   17804 Fri Dec 21 14:59:53 2018
new/exception_lists/hdrchk
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 usr/src/boot/*
   2 usr/src/cmd/acpi/acpidump/acpidump.h
   3 usr/src/cmd/acpi/acpixtract/acpixtract.h
   4 usr/src/cmd/acpi/acpixtract/axmain.h
   5 usr/src/cmd/acpi/acpixtract/axutils.h
   6 usr/src/cmd/cmd-inet/usr.bin/dns-sd/ClientCommon.h
   7 usr/src/cmd/cmd-inet/usr.lib/mdnsd/CryptoAlg.h
   8 usr/src/cmd/cmd-inet/usr.lib/mdnsd/DNSCommon.h
   9 usr/src/cmd/cmd-inet/usr.lib/mdnsd/GenLinkedList.h
  10 usr/src/cmd/cmd-inet/usr.lib/mdnsd/PlatformCommon.h
  11 usr/src/cmd/cmd-inet/usr.lib/mdnsd/anonymous.h
  12 usr/src/cmd/cmd-inet/usr.lib/mdnsd/dnssec.h
  13 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSDebug.h
  14 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSEmbeddedAPI.h
  15 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSPosix.h
  16 usr/src/cmd/cmd-inet/usr.lib/mdnsd/mDNSUNP.h
  17 usr/src/cmd/cmd-inet/usr.lib/mdnsd/uDNS.h
  18 usr/src/cmd/cmd-inet/usr.lib/mdnsd/nsec.h
  19 usr/src/cmd/cmd-inet/usr.lib/mdnsd/uds_daemon.h
  20 usr/src/cmd/krb5/kadmin/cli/kadmin.h
  21 usr/src/cmd/krb5/kadmin/dbutil/import_err.h
  22 usr/src/cmd/krb5/kadmin/dbutil/kdb5_util.h
  23 usr/src/cmd/krb5/kadmin/dbutil/nstrtok.h
  24 usr/src/cmd/krb5/kadmin/dbutil/string_table.h
  25 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd_strings.h
  26 usr/src/cmd/krb5/kadmin/kpasswd/kpasswd.h
  27 usr/src/cmd/krb5/kadmin/ktutil/ktutil.h
  28 usr/src/cmd/krb5/kadmin/server/misc.h
  29 usr/src/cmd/krb5/krb5kdc/extern.h
  30 usr/src/cmd/krb5/krb5kdc/kdc_util.h
  31 usr/src/cmd/krb5/krb5kdc/policy.h
  32 usr/src/cmd/krb5/ldap_util/kdb5_ldap_list.h
  33 usr/src/cmd/krb5/ldap_util/kdb5_ldap_policy.h
  34 usr/src/cmd/krb5/ldap_util/kdb5_ldap_realm.h
  35 usr/src/cmd/krb5/ldap_util/kdb5_ldap_services.h
  36 usr/src/cmd/krb5/ldap_util/kdb5_ldap_util.h
  37 usr/src/cmd/krb5/slave/kprop.h
  38 usr/src/cmd/localedef/localedef.h
  39 usr/src/cmd/mandoc
  40 usr/src/common/ficl/ficltokens.h
  41 usr/src/grub/grub-0.97/stage2/shared.h
  42 usr/src/lib/gss_mechs/mech_krb5/et/error_table.h
  43 usr/src/lib/gss_mechs/mech_krb5/et/internal.h
  44 usr/src/lib/gss_mechs/mech_krb5/et/mit-sipb-copyright.h
  45 usr/src/lib/gss_mechs/mech_krb5/include/cache-addrinfo.h
  46 usr/src/lib/gss_mechs/mech_krb5/include/cm.h
  47 usr/src/lib/gss_mechs/mech_krb5/include/com_err.h
  48 usr/src/lib/gss_mechs/mech_krb5/include/db-config.h
  49 usr/src/lib/gss_mechs/mech_krb5/include/db.h
  50 usr/src/lib/gss_mechs/mech_krb5/include/fake-addrinfo.h
  51 usr/src/lib/gss_mechs/mech_krb5/include/foreachaddr.h
  52 usr/src/lib/gss_mechs/mech_krb5/include/k5-int-pkinit.h
  53 usr/src/lib/gss_mechs/mech_krb5/include/k5-utf8.h
  54 usr/src/lib/gss_mechs/mech_krb5/include/kdb_kt.h
  55 usr/src/lib/gss_mechs/mech_krb5/include/krb5_libinit.h
  56 usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_defs.h
  57 usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm_proto.h
  58 usr/src/lib/gss_mechs/mech_krb5/include/krb5/adm.h
  59 usr/src/lib/gss_mechs/mech_krb5/include/krb5/copyright.h
  60 usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-err.h
```



```
  61 usr/src/lib/gss_mechs/mech_krb5/include/krb5/k5-plugin.h
  62 usr/src/lib/gss_mechs/mech_krb5/include/krb5/kdb_dbc.h
  63 usr/src/lib/gss_mechs/mech_krb5/include/krb5/kdb.h
  64 usr/src/lib/gss_mechs/mech_krb5/include/locate_plugin.h
  65 usr/src/lib/gss_mechs/mech_krb5/include/osconf.h
  66 usr/src/lib/gss_mechs/mech_krb5/include/port-sockets.h
  67 usr/src/lib/gss_mechs/mech_krb5/include/preauth_plugin.h
  68 usr/src/lib/gss_mechs/mech_krb5/include/socket-utils.h
  69 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_decode.h
  70 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_encode.h
  71 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_get.h
  72 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_decode.h
  73 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_k_encode.h
  74 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_make.h
  75 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1_misc.h
  76 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/asn1buf.h
  77 usr/src/lib/gss_mechs/mech_krb5/krb5/asn.1/krbasn1.h
  78 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/cc-int.h
  79 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/fcc.h
  80 usr/src/lib/gss_mechs/mech_krb5/krb5/ccache/scc.h
  81 usr/src/lib/gss_mechs/mech_krb5/krb5/error_tables/adm_err.h
  82 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/file/ktfile.h
  83 usr/src/lib/gss_mechs/mech_krb5/krb5/keytab/kt-int.h
  84 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/cleanup.h
  85 usr/src/lib/gss_mechs/mech_krb5/krb5/krb/int-proto.h
  86 usr/src/lib/gss_mechs/mech_krb5/krb5/os/dnsglue.h
  87 usr/src/lib/gss_mechs/mech_krb5/krb5/os/os-proto.h
  88 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_base.h
  89 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc_io.h
  90 usr/src/lib/gss_mechs/mech_krb5/krb5/rcache/rc-int.h
  91 usr/src/lib/gss_mechs/mech_krb5/mech/gss_libinit.h
  92 usr/src/lib/gss_mechs/mech_krb5/profile/prof_err.h
  93 usr/src/lib/gss_mechs/mech_krb5/support/supp-int.h
  94 usr/src/lib/krb5/kadm5/admin_internal.h
  95 usr/src/lib/krb5/kadm5/admin_xdr.h
  96 usr/src/lib/krb5/kadm5/admin.h
  97 usr/src/lib/krb5/kadm5/chpass_util_strings.h
  98 usr/src/lib/krb5/kadm5/clnt/client_internal.h
  99 usr/src/lib/krb5/kadm5/kadm_err.h
 100 usr/src/lib/krb5/kadm5/kadm_rpc.h
 101 usr/src/lib/krb5/kadm5/server_internal.h
 102 usr/src/lib/krb5/kadm5/srv/server_acl.h
 103 usr/src/lib/krb5/kdb/adb_err.h
 104 usr/src/lib/krb5/kdb/kdb5.h
 105 usr/src/lib/krb5/plugins/kdb/db2/kdb_compat.h
 106 usr/src/lib/krb5/plugins/kdb/db2/kdb_db2.h
 107 usr/src/lib/krb5/plugins/kdb/db2/kdb_xdr.h
 108 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/btree.h
 109 usr/src/lib/krb5/plugins/kdb/db2/libdb2/btree/extern.h
 110 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/extern.h
 111 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/hash.h
 112 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/page.h
 113 usr/src/lib/krb5/plugins/kdb/db2/libdb2/hash/search.h
 114 usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-int.h
 115 usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-ndbm.h
 116 usr/src/lib/krb5/plugins/kdb/db2/libdb2/include/db-queue.h
 117 usr/src/lib/krb5/plugins/kdb/db2/libdb2/mpool/mpool.h
 118 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/extern.h
 119 usr/src/lib/krb5/plugins/kdb/db2/libdb2/recno/recno.h
 120 usr/src/lib/krb5/plugins/kdb/db2/policy_db.h
 121 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_ldap.h
 122 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/kdb_xdr.h
 123 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_err.h
 124 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_handle.h
 125 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_krbcontainer.h
 126 usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_main.h
```

```
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_misc.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_principal.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_pwd_policy.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_realm.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_service_stash.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_services.h
usr/src/lib/krb5/plugins/kdb/ldap/libkdb_ldap/ldap_tkt_policy.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_accessor.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto_openssl.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit_crypto.h
usr/src/lib/krb5/plugins/preauth/pkinit/pkinit.h
usr/src/lib/krb5/ss/copyright.h
usr/src/lib/krb5/ss/mit-sipb-copyright.h
usr/src/lib/krb5/ss/ss_internal.h
usr/src/lib/krb5/ss/ss.h
usr/src/lib/libc/port/locale/utils.h
usr/src/lib/libdns_sd/common/dns_sd.h
usr/src/lib/libdns_sd/common/dnssd_ipc.h
usr/src/lib/librstp/common/base.h
usr/src/lib/librstp/common/choose.h
usr/src/lib/librstp/common/edge.h
usr/src/lib/librstp/common/migrate.h
usr/src/lib/librstp/common/p2p.h
usr/src/lib/librstp/common/pcost.h
usr/src/lib/librstp/common/port.h
usr/src/lib/librstp/common/portinfo.h
usr/src/lib/librstp/common/rolesel.h
usr/src/lib/librstp/common/roletrns.h
usr/src/lib/librstp/common/statmch.h
usr/src/lib/librstp/common/stp_bpdu.h
usr/src/lib/librstp/common/stp_in.h
usr/src/lib/librstp/common/stp_to.h
usr/src/lib/librstp/common/stp_vectors.h
usr/src/lib/librstp/common/stpm.h
usr/src/lib/librstp/common/sttrans.h
usr/src/lib/librstp/common/times.h
usr/src/lib/librstp/common/topoch.h
usr/src/lib/librstp/common/transmit.h
usr/src/lib/librstp/common/uid_stp.h
usr/src/lib/librstp/common/vector.h
usr/src/lib/libsmbfs/netsmb/spnego.h
usr/src/lib/libsmbfs/smb/derparse.h
usr/src/lib/libsmbfs/smb/spnegoparse.h
usr/src/tools/btxld/*
usr/src/tools/smatch/src/*
usr/src/uts/common/gssapi/mechs/krb5/include/aes_s2k.h
usr/src/uts/common/gssapi/mechs/krb5/include/auth_con.h
usr/src/uts/common/gssapi/mechs/krb5/include/cksumtypes.h
usr/src/uts/common/gssapi/mechs/krb5/include/crc-32.h
usr/src/uts/common/gssapi/mechs/krb5/include/des_int.h
usr/src/uts/common/gssapi/mechs/krb5/include/dk.h
usr/src/uts/common/gssapi/mechs/krb5/include/enc_provider.h
usr/src/uts/common/gssapi/mechs/krb5/include/etypes.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_generic.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapi_krb5.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_generic.h
usr/src/uts/common/gssapi/mechs/krb5/include/gssapiP_krb5.h
usr/src/uts/common/gssapi/mechs/krb5/include/hash_provider.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-int.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_16.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_32.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-load_64.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_16.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_32.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform-store_64.h
usr/src/uts/common/gssapi/mechs/krb5/include/k5-platform.h
```



```
usr/src/uts/common/gssapi/mechs/krb5/include/k5-thread.h
usr/src/uts/common/gssapi/mechs/krb5/include/keyhash_provider.h
usr/src/uts/common/gssapi/mechs/krb5/include/krb5.h
usr/src/uts/common/gssapi/mechs/krb5/include/old.h
usr/src/uts/common/gssapi/mechs/krb5/include/raw.h
usr/src/uts/common/gssapi/mechs/krb5/include/rsa-md4.h
usr/src/uts/common/io/axf/ax88172reg.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_common.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_init_ops.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_init.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/ecore/ecore_sp_verbs.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/bcm_utils.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/cyclic_oper.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/debug.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/l4/mm_l4if.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/l5/mm_l5if.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_dos.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_linux.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_ndismono.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_solaris.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_uefi.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_user_mode_debug.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm_vbd.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/mm.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/utils.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/vm/hw_channel.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/include/vm/vfpf_if.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/bd_chain_st.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/bd_chain.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/context.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/hw_debug.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_dcbx_mp.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_desc.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_dmae.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/device/lm_sp_req_mgr.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/5710_hsi.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57710_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57711_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57712_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/577xx_int_offsets.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_fcoe_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_fcoe_rfc_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_iscsi_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_iscsi_rfc_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/57xx_l5cm_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/eth_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/everest_iscsi_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/everest_l5cm_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/fcoe_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/fw_defs.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/init_defs.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/microcode_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/rdma_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/tcp_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/fw/toe_constants.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/command.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/common_uif.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm_stats.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm5710_hsi.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/include/lm5710.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/include/lm_l4if.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/include/lm_l4st.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4fp.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l4/lm_l4sp.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l5/include/lm_l5if.h
```

```
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/l5/include/lm_l5st.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/basic_vf/lm_vf.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/channel_vf/lm_vf.h
usr/src/uts/common/io/bnxe/577xx/drivers/common/lm/vf/common/lm_vf_common.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/57712_reg.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/aeu_inputs.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/bigmac_addresses.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/cdu_def.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/clc_reg.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/clc.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/dbg_bus.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/dbu_reg_Driver.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/dmae_clients.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/emac_reg_driver.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/general_atten_bits.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/grc_addr.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/hw_dump.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/igu_def.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/misc_bits.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/pcics_reg_driver.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/phy_reg.h
usr/src/uts/common/io/bnxe/577xx/hsi/hw/include/prs_flags.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/append.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/bdn.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/code_swap_def.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/dev_info.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mac_drv_info.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mac_stats.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mac_stx.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mcp_fio.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/mcp_shmem.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/multi_thread_def.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/ncsi_basic_types.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/ncsi_cmds.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/nvm_map.h
usr/src/uts/common/io/bnxe/577xx/hsi/mcp/shmem.h
usr/src/uts/common/io/bnxe/577xx/include/bcmtype.h
usr/src/uts/common/io/bnxe/577xx/include/iscsi_info.h
usr/src/uts/common/io/bnxe/577xx/include/l4debug.h
usr/src/uts/common/io/bnxe/577xx/include/l4states.h
usr/src/uts/common/io/bnxe/577xx/include/license.h
usr/src/uts/common/io/bnxe/577xx/include/listq.h
usr/src/uts/common/io/bnxe/577xx/include/lm_defs.h
usr/src/uts/common/io/bnxe/bnxe_binding.h
usr/src/uts/common/io/bnxe/bnxe_debug.h
usr/src/uts/common/io/bnxe/bnxe.h
usr/src/uts/common/io/bnxe/version.h
usr/src/uts/common/io/cxgbe/firmware/*
usr/src/uts/common/io/cxgbe/common/t4_msg.h
usr/src/uts/common/io/cxgbe/common/t4_regs.h
usr/src/uts/common/io/e1000api/e1000_80003es2lan.h
usr/src/uts/common/io/e1000api/e1000_82541.h
usr/src/uts/common/io/e1000api/e1000_82543.h
usr/src/uts/common/io/e1000api/e1000_82571.h
usr/src/uts/common/io/e1000api/e1000_hw.h
usr/src/uts/common/io/e1000api/e1000_i210.h
usr/src/uts/common/io/e1000api/e1000_mac.h
usr/src/uts/common/io/e1000api/e1000_manage.h
usr/src/uts/common/io/e1000api/e1000_nvm.h
usr/src/uts/common/io/e1000api/e1000_phy.h
usr/src/uts/common/io/e1000api/e1000_regs.h
usr/src/uts/common/io/ixgbe/core/ixgbe_common.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82598.h
usr/src/uts/common/io/ixgbe/core/ixgbe_dcb_82599.h
usr/src/uts/common/io/i40e/core/i40e_adminq_cmd.h
usr/src/uts/common/io/qede/*
```



```
usr/src/uts/common/io/udmf/dm9601reg.h
usr/src/uts/common/io/upf/adm8511reg.h
usr/src/uts/common/io/urf/rtl8150reg.h
usr/src/uts/common/sys/scsi/adapters/mpt_sas/mpi/*
usr/src/uts/intel/sys/acpi/acdebug.h
usr/src/uts/intel/sys/acpi/acdisasm.h
usr/src/uts/intel/sys/acpi/acevents.h
usr/src/uts/intel/sys/acpi/acinterp.h
usr/src/uts/intel/sys/acpi/acmacros.h
usr/src/uts/intel/sys/acpi/acnames.h
usr/src/uts/intel/sys/acpi/acpredef.h
usr/src/uts/intel/sys/acpi/acresrc.h
usr/src/uts/intel/sys/acpi/acstruct.h
usr/src/uts/intel/sys/acpi/amlresrc.h
usr/src/uts/intel/sys/acpi/platform/acwin64.h
```

```
***********************************************************
     955 Fri Dec 21 14:59:53 2018
new/exception_lists/manlint
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
***********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #
  11 #
  12 # Copyright 2016 Toomas Soome <tsoome@me.com>
  13 # Copyright (c) 2018, Joyent, Inc.
  14 #
  15 usr/src/boot/*
  16 # Not actually a manual page
  17 usr/src/cmd/hal/fdi/fdi.dtd.1
  18 usr/src/cmd/isns/isnsd/xml_def/isnsdata.dtd.1
  19 usr/src/cmd/svc/dtd/service_bundle.dtd.1
  20 usr/src/lib/fm/topo/maps/common/topology.dtd.1
  21 usr/src/lib/libbrand/dtd/brand.dtd.1
  22 usr/src/lib/libbrand/dtd/zone_platform.dtd.1
  23 usr/src/lib/libbsm/adt_record.dtd.1
  24 usr/src/lib/libbsm/adt_record.xsl.1
  25 usr/src/lib/libpool/dtd/rm_pool.dtd.1
  26 usr/src/lib/libzonecfg/dtd/zonecfg.dtd.1
  27 usr/src/tools/smatch/src/*
```

```
**********************************************************
   28340 Fri Dec 21 14:59:54 2018
new/exception_lists/packaging
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright 2012 OmniTI Computer Consulting, Inc.  All rights reserved.
  25 # Copyright 2014 Garrett D'Amore <garrett@damore.org>
  26 # Copyright 2016 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
  27 # Copyright 2018 Nexenta Systems, Inc.
  28 # Copyright 2017 Toomas Soome <tsoome@me.com>
  29 # Copyright 2017 RackTop Systems.
  30 # Copyright 2018, Joyent, Inc.
  31 # Copyright 2018 Jason King
  32 #

  34 #
  35 # Exception List for validate_pkg
  36 #

  38 #
  39 # The following entries are built in the /proto area
  40 # but not included in any packages - this is intentional.
  41 #
  42 usr/include/auth_list.h
  43 usr/include/bsm/audit_door_infc.h
  44 usr/include/bsm/audit_private.h
  45 usr/include/bsm/devalloc.h
  46 usr/include/demangle-sys.h
  47 usr/include/getxby_door.h
  48 usr/include/passwdutil.h
  49 usr/include/priv_utils.h
  50 usr/include/rpcsvc/daemon_utils.h
  51 usr/include/rpcsvc/svc_dg_priv.h
  52 usr/include/security/pam_impl.h
  53 usr/include/sys/clock_impl.h
  54 usr/include/sys/winlockio.h
  55 usr/include/scsi/plugins/ses/vendor/sun_impl.h
  56 #
  57 # Private lofi interface.
  58 #
  59 usr/include/sys/lofi_impl.h
  60 #
```



```
  61 # Private/Internal libraries of the Cryptographic Framework.
  62 #
  63 lib/libkcfd.so
  64 lib/llib-lelfsign
  65 lib/llib-lelfsign.ln
  66 lib/llib-lkcfd
  67 lib/llib-lkcfd.ln
  68 usr/include/libelfsign.h
  69 usr/lib/llib-lsoftcrypto
  70 usr/lib/llib-lsoftcrypto.ln
  71 usr/lib/amd64/llib-lsoftcrypto.ln       i386
  72 usr/lib/sparcv9/llib-lsoftcrypto.ln     sparc

  74 #
  75 # The following files are used by the DHCP service, the
  76 # standalone's DHCP implementation, and the kernel (nfs_dlboot).
  77 # They contain interfaces which are currently private.
  78 #
  79 usr/include/dhcp_symbol.h
  80 usr/include/sys/sunos_dhcp_class.h
  81 #
  82 # Private MAC driver header files
  83 #
  84 usr/include/inet/iptun.h
  85 usr/include/sys/aggr_impl.h
  86 usr/include/sys/aggr.h
  87 usr/include/sys/dld_impl.h
  88 usr/include/sys/dld_ioc.h
  89 usr/include/sys/dls_impl.h
  90 usr/include/sys/dls.h
  91 usr/include/sys/mac_client_impl.h
  92 usr/include/sys/mac_client.h
  93 usr/include/sys/mac_flow_impl.h
  94 usr/include/sys/mac_impl.h
  95 usr/include/sys/mac_soft_ring.h
  96 usr/include/sys/mac_stat.h
  97 #
  98 # Private GLDv3 userland libraries and headers
  99 #
 100 usr/include/libdladm_impl.h
 101 usr/include/libdlaggr.h
 102 usr/include/libdlether.h
 103 usr/include/libdlflow_impl.h
 104 usr/include/libdlflow.h
 105 usr/include/libdliptun.h
 106 usr/include/libdlmgmt.h
 107 usr/include/libdlsim.h
 108 usr/include/libdlstat.h
 109 usr/include/libdlvnic.h
 110 usr/include/libdlwlan_impl.h
 111 usr/include/libdlwlan.h
 112 #
 113 # Virtual Network Interface Card (VNIC)
 114 #
 115 usr/include/sys/vnic.h
 116 usr/include/sys/vnic_impl.h
 117 #
 118 # Private libipadm lint library and header files
 119 #
 120 usr/include/ipadm_ipmgmt.h
 121 usr/include/ipadm_ndpd.h
 122 usr/include/libipadm.h
 123 lib/llib-lipadm
 124 lib/llib-lipadm.ln
 125 lib/libipadm.so
 126 #
```

```
127 # Private libsocket header file
128 #
129 usr/include/libsocket_priv.h
130 #
131 # IKE and IPsec support library exceptions.  The IKE support
132 # library contains exclusively private interfaces, as does
133 # libipsecutil.  My apologies for the glut of header files here.
134 #
135 usr/include/errfp.h
136 usr/include/ikedoor.h
137 usr/include/ipsec_util.h
138 usr/lib/amd64/libike.so                 i386
139 usr/lib/sparcv9/libike.so               sparc
140 usr/lib/libipsecutil.so
141 usr/lib/amd64/libipsecutil.so           i386
142 usr/lib/sparcv9/libipsecutil.so         sparc
143 usr/lib/llib-like
144 usr/lib/llib-like.ln
145 usr/lib/amd64/llib-like.ln              i386
146 usr/lib/sparcv9/llib-like.ln            sparc
147 usr/lib/llib-lipsecutil
148 usr/lib/llib-lipsecutil.ln
149 usr/lib/amd64/llib-lipsecutil.ln        i386
150 usr/lib/sparcv9/llib-lipsecutil.ln      sparc
151 #
152 usr/include/inet/ip_impl.h
153 usr/include/inet/ip_ndp.h
154 usr/include/inet/ip2mac_impl.h
155 usr/include/inet/ip2mac.h
156 usr/include/inet/rawip_impl.h
157 usr/include/inet/tcp_impl.h
158 usr/include/inet/udp_impl.h
159 usr/include/libmail.h
160 usr/include/libnwam_priv.h
161 usr/include/protocols/ripngd.h
162 usr/include/s_string.h
163 usr/include/sys/logindmux_impl.h
164 usr/include/sys/vgareg.h
165 #
166 # Some IPsec headers can’t be shipped lest we hit export controls...
167 #
168 usr/include/inet/ipsec_impl.h
169 usr/include/inet/ipsec_info.h
170 usr/include/inet/ipsecah.h
171 usr/include/inet/ipsecesp.h
172 usr/include/inet/keysock.h
173 usr/include/inet/sadb.h
174 usr/include/sys/sha1_consts.h
175 usr/include/sys/sha2_consts.h
176 #
177 #
178 # Filtering out directories not shipped
179 #
180 usr/4lib                                i386
181 #
182 # These files contain definitions shared privately between the kernel
183 # and libc.  There is no reason for them to be part of a package that
184 # a customer should ever see.  They are installed in the proto area by
185 # the uts build because libc and and other components, like truss, are
186 # dependent upon their contents and should not have their own copies.
187 #
188 usr/include/sys/libc_kernel.h
189 usr/include/sys/synch32.h
190 #
191 # Private interfaces for libdisasm
192 #
```



```
193 usr/include/libdisasm.h
194 usr/lib/llib-ldisasm
195 usr/lib/llib-ldisasm.ln
196 usr/lib/amd64/llib-ldisasm.ln           i386
197 usr/lib/sparcv9/llib-ldisasm.ln         sparc
198 #
199 # Private interfaces for libraidcfg
200 #
201 usr/include/raidcfg_spi.h
202 usr/include/raidcfg.h
203 usr/lib/libraidcfg.so
204 usr/lib/amd64/libraidcfg.so             i386
205 usr/lib/sparcv9/libraidcfg.so           sparc
206 usr/lib/llib-lraidcfg
207 usr/lib/llib-lraidcfg.ln
208 usr/lib/amd64/llib-lraidcfg.ln          i386
209 usr/lib/sparcv9/llib-lraidcfg.ln        sparc
210 #
211 # This file is used for private communication between mdb, drv/kmdb, and
212 # misc/kmdb.  The interfaces described herein are not intended for customer
213 # use, and are thus excluded from packaging.
214 #
215 usr/include/sys/kmdb.h
216 #
217 # These files are installed in the proto area by the build of libdhcpagent
218 # and libdhcputil for the benefit of DHCP-related networking commands such
219 # as dhcpagent, dhcpinfo, ifconfig, and netstat.  These are not interfaces
220 # for customer use, so the files are excluded from packaging.
221 #
222 lib/libdhcpagent.so
223 lib/libdhcputil.so
224 lib/amd64/libdhcputil.so                i386
225 lib/sparcv9/libdhcputil.so              sparc
226 lib/llib-ldhcpagent
227 lib/llib-ldhcpagent.ln
228 lib/llib-ldhcputil
229 lib/llib-ldhcputil.ln
230 lib/amd64/llib-ldhcputil.ln             i386
231 lib/sparcv9/llib-ldhcputil.ln           sparc
232 usr/include/dhcp_hostconf.h
233 usr/include/dhcp_impl.h
234 usr/include/dhcp_inittab.h
235 usr/include/dhcp_stable.h
236 usr/include/dhcp_symbol_common.h
237 usr/include/dhcpagent_ipc.h
238 usr/include/dhcpagent_util.h
239 usr/include/dhcpmsg.h
240 usr/lib/libdhcpagent.so
241 usr/lib/libdhcputil.so
242 usr/lib/amd64/libdhcputil.so            i386
243 usr/lib/sparcv9/libdhcputil.so          sparc
244 usr/lib/llib-ldhcpagent
245 usr/lib/llib-ldhcpagent.ln
246 usr/lib/llib-ldhcputil
247 usr/lib/llib-ldhcputil.ln
248 usr/lib/amd64/llib-ldhcputil.ln         i386
249 usr/lib/sparcv9/llib-ldhcputil.ln       sparc
250 #
251 # These files are installed in the proto area by the build of libinstzones
252 # and libpkg
253 #
254 usr/lib/llib-linstzones
255 usr/lib/llib-linstzones.ln
256 usr/lib/amd64/llib-linstzones.ln        i386
257 usr/lib/sparcv9/llib-linstzones.ln      sparc
258 usr/lib/llib-lpkg
```

```
usr/lib/llib-lpkg.ln
#
# Don’t ship header files private to libipmp and in.mpathd
#
usr/include/ipmp_query_impl.h
#
# These files are installed in the proto area by the build of libinetsvc,
# an inetd-specific library shared by inetd, inetadm and inetconv. Only
# the shared object is shipped.
#
usr/include/inetsvc.h
usr/lib/libinetsvc.so
usr/lib/llib-linetsvc
usr/lib/llib-linetsvc.ln
#
# These files are installed in the proto area by the build of libinetutil,
# a general purpose library for the benefit of internet utilities. Only
# the shared object is shipped.
#
lib/libinetutil.so
lib/amd64/libinetutil.so                i386
lib/sparcv9/libinetutil.so              sparc
lib/llib-linetutil
lib/llib-linetutil.ln
lib/amd64/llib-linetutil.ln             i386
lib/sparcv9/llib-linetutil.ln           sparc
usr/include/libinetutil.h
usr/include/netinet/inetutil.h
usr/include/ofmt.h
usr/lib/libinetutil.so
usr/lib/amd64/libinetutil.so            i386
usr/lib/sparcv9/libinetutil.so          sparc
usr/lib/llib-linetutil
usr/lib/llib-linetutil.ln
usr/lib/amd64/llib-linetutil.ln         i386
usr/lib/sparcv9/llib-linetutil.ln       sparc
#
# Miscellaneous kernel interfaces or kernel<->user interfaces that are
# consolidation private and we do not want to export at this time.
#
usr/include/sys/cryptmod.h
usr/include/sys/dumpadm.h
usr/include/sys/ontrap.h
usr/include/sys/sysmsg_impl.h
usr/include/sys/vlan.h
#
# non-public pci header
#
usr/include/sys/pci_impl.h
usr/include/sys/pci_tools.h
#
# Exception list for RCM project, included by librcm and rcm_daemon
#
usr/include/librcm_event.h
usr/include/librcm_impl.h
#
# MDB deliverables that are not yet public
#
usr/lib/mdb/proc/mdb_test.so
usr/lib/mdb/proc/sparcv9/mdb_test.so    sparc
#
# SNCA project exception list
#
usr/include/inet/kssl/kssl.h
usr/include/inet/kssl/ksslimpl.h
usr/include/inet/kssl/ksslproto.h
```



```
usr/include/inet/nca
#
# these are "removed" from the source product build because the only
# packages that currently deliver them are removed.
# they really should’t be in here.
#
etc/sfw
#
# Entries for the libmech_krb5 symlink, which has been included
# for build purposes only, not delivered to customers.
#
usr/include/gssapi/gssapi_krb5.h
usr/lib/gss/libmech_krb5.so
usr/lib/amd64/gss/libmech_krb5.so       i386
usr/lib/sparcv9/gss/libmech_krb5.so     sparc
usr/lib/libmech_krb5.so
usr/lib/amd64/libmech_krb5.so           i386
usr/lib/sparcv9/libmech_krb5.so         sparc
#
# Entries for headers from efcode project which user does not need to see
#
usr/platform/sun4u/include/sys/fc_plat.h                        sparc
usr/platform/sun4u/include/sys/fcode.h                          sparc
#
# Private net80211 headers
#
usr/include/sys/net80211_amrr.h
usr/include/sys/net80211_crypto.h
usr/include/sys/net80211_ht.h
usr/include/sys/net80211_proto.h
usr/include/sys/net80211.h
#
usr/include/net/wpa.h
#
# PPPoE files not delivered to customers.
#
usr/include/net/pppoe.h
usr/include/net/sppptun.h
#
# Simnet
#
usr/include/net/simnet.h
#
# Bridging internal data structures
#
usr/include/net/bridge_impl.h
#
# User<->kernel interface used by cfgadm/USB only
#
usr/include/sys/usb/hubd/hubd_impl.h
#
# User<->kernel interface used by cfgadm/SATA only
#
usr/include/sys/sata/sata_cfgadm.h                      i386
#
# Private ucred kernel header
#
usr/include/sys/ucred.h
#
# Private and/or platform-specific smf(5) files
#
lib/librestart.so
lib/llib-lrestart
lib/llib-lrestart.ln
lib/amd64/llib-lrestart.ln                                      i386
lib/sparcv9/llib-lrestart.ln                                    sparc
```

```
usr/include/libcontract_priv.h
usr/include/librestart_priv.h
usr/include/librestart.h
usr/lib/librestart.so
usr/lib/sparcv9/librestart.so                                   sparc
lib/svc/manifest/platform/sun4u                                 i386
lib/svc/manifest/platform/sun4v                                 i386
var/svc/manifest/platform/sun4u                                 i386
var/svc/manifest/platform/sun4v                                 i386
etc/svc/profile/platform_sun4v.xml                              i386
etc/svc/profile/platform_SUNW,SPARC-Enterprise.xml              i386
etc/svc/profile/platform_SUNW,Sun-Fire-15000.xml                i386
etc/svc/profile/platform_SUNW,Sun-Fire-880.xml                  i386
etc/svc/profile/platform_SUNW,Sun-Fire-V890.xml                 i386
etc/svc/profile/platform_SUNW,Sun-Fire.xml                      i386
etc/svc/profile/platform_SUNW,Ultra-Enterprise-10000.xml        i386
etc/svc/profile/platform_SUNW,UltraSPARC-IIe-NetraCT-40.xml     i386
etc/svc/profile/platform_SUNW,UltraSPARC-IIe-NetraCT-60.xml     i386
etc/svc/profile/platform_SUNW,UltraSPARC-IIi-Netract.xml        i386
#
# Private libuutil files
#
lib/libuutil.so
lib/llib-luutil
lib/llib-luutil.ln
lib/sparcv9/llib-luutil.ln              sparc
usr/include/libuutil_impl.h
usr/lib/libuutil.so
usr/lib/sparcv9/libuutil.so             sparc
#
# Private Multidata file.
#
usr/include/sys/multidata_impl.h
#
# Even though all the objects built under usr/src/stand are later glommed
# together into a couple of second-stage boot loaders, we dump the static
# archives and lint libraries into $(ROOT)/stand for intermediate use
# (e.g., for lint, linking the second-stage boot loaders, ...).  Since
# these are merely intermediate objects, they do not need to be packaged.
#
stand                                   sparc
#
# Private KCF header files
#
usr/include/sys/crypto/elfsign.h
usr/include/sys/crypto/impl.h
usr/include/sys/crypto/ops_impl.h
usr/include/sys/crypto/sched_impl.h
#
# The following files are installed in the proto area
# by the build of libcmdutils (Command Utilities Library).
# libcmdutils contains interfaces which are all private interfaces.
#
lib/libcmdutils.so
lib/amd64/libcmdutils.so                i386
lib/sparcv9/libcmdutils.so              sparc
lib/llib-lcmdutils
lib/llib-lcmdutils.ln
lib/amd64/llib-lcmdutils.ln             i386
lib/sparcv9/llib-lcmdutils.ln           sparc
usr/include/libcmdutils.h
usr/lib/libcmdutils.so
usr/lib/amd64/libcmdutils.so            i386
usr/lib/sparcv9/libcmdutils.so          sparc
usr/lib/llib-lcmdutils
usr/lib/llib-lcmdutils.ln
```



```
usr/lib/amd64/llib-lcmdutils.ln         i386
usr/lib/sparcv9/llib-lcmdutils.ln       sparc
#
# Private interfaces in libsec
#
usr/include/aclutils.h
#
# USB skeleton driver stays in sync with the rest of USB but doesn't ship.
#
kernel/drv/amd64/usbskel        i386
kernel/drv/sparcv9/usbskel      sparc
kernel/drv/usbskel.conf
#
# Consolidation and Sun private libdevid interfaces
# Public libdevid interfaces provided by devid.h
#
usr/include/sys/libdevid.h
#
# The following files are installed in the proto area by the build of
# libprtdiag.  libprtdiag contains interfaces which are all private.
# Only the shared object is shipped.
#
usr/platform/sun4u/lib/llib-lprtdiag                            sparc
usr/platform/sun4u/lib/llib-lprtdiag.ln                         sparc
usr/platform/sun4v/lib/llib-lprtdiag.ln                         sparc
#
# The following files are installed in the proto area by the build of
# mdesc driver in sun4v. These header files are used on in the build
# and do not need to be shipped to customers.
#
usr/include/sys/mdesc.h                                 sparc
usr/include/sys/mdesc_impl.h                            sparc
usr/platform/sun4v/include/sys/mach_descrip.h           sparc
#
# The following files are installed in the proto area by the build of
# libpcp.  libpcp contains interfaces which are all private.
# Only the shared object is shipped.
#
usr/platform/sun4v/lib/llib-lpcp.ln                             sparc
usr/platform/SUNW,Netra-CP3060/lib/llib-lpcp.ln                 sparc
usr/platform/SUNW,Netra-CP3260/lib/llib-lpcp.ln                 sparc
usr/platform/SUNW,Netra-T5220/lib/llib-lpcp.ln                  sparc
usr/platform/SUNW,Netra-T5440/lib/llib-lpcp.ln                  sparc
usr/platform/SUNW,SPARC-Enterprise-T5120/lib/llib-lpcp.ln       sparc
usr/platform/SUNW,Sun-Blade-T6300/lib/llib-lpcp.ln              sparc
usr/platform/SUNW,Sun-Blade-T6320/lib/llib-lpcp.ln              sparc
usr/platform/SUNW,Sun-Fire-T200/lib/llib-lpcp.ln                sparc
usr/platform/SUNW,T5140/lib/llib-lpcp.ln                        sparc
usr/platform/SUNW,USBRDT-5240/lib/llib-lpcp.ln                  sparc
#
# ZFS internal tools and lint libraries
#
usr/lib/llib-lzfs_jni
usr/lib/llib-lzfs_jni.ln
usr/lib/amd64/llib-lzfs_jni.ln          i386
usr/lib/sparcv9/llib-lzfs_jni.ln        sparc
usr/lib/llib-lzpool
usr/lib/llib-lzpool.ln                  i386
usr/lib/amd64/llib-lzpool.ln            i386
usr/lib/sparcv9/llib-lzpool.ln          sparc
#
# ZFS JNI headers
#
usr/include/libzfs_jni_dataset.h
usr/include/libzfs_jni_disk.h
usr/include/libzfs_jni_diskmgt.h
```

```
usr/include/libzfs_jni_ipool.h
usr/include/libzfs_jni_main.h
usr/include/libzfs_jni_pool.h
usr/include/libzfs_jni_property.h
usr/include/libzfs_jni_util.h
#
# These files are installed in the proto area for Solaris scsi_vhci driver
# (for MPAPI support) and should not be shipped
#
usr/include/sys/scsi/adapters/mpapi_impl.h
usr/include/sys/scsi/adapters/mpapi_scsi_vhci.h
#
# This library is installed in the proto area by the build of libdisasm, and is
# only used when building the KMDB disasm module.
#
usr/lib/libstanddisasm.so
usr/lib/amd64/libstanddisasm.so         i386
usr/lib/sparcv9/libstanddisasm.so       sparc
#
# TSol: tsol doesn't ship lint source, and tsnet isn't for customers at all.
#
lib/libtsnet.so
usr/lib/llib-ltsnet
usr/lib/llib-ltsol
#
# nss interfaces shared between libnsl and other ON libraries.
#
usr/include/nss.h
#
# AT&T AST (ksh93) files which are currently needed only to build OS/Net
# (msgcc&co.)
# libast
usr/lib/libast.so
usr/lib/amd64/libast.so                 i386
usr/lib/sparcv9/libast.so               sparc
usr/lib/llib-last
usr/lib/llib-last.ln
usr/lib/amd64/llib-last.ln              i386
usr/lib/sparcv9/llib-last.ln            sparc
# libcmd
usr/lib/llib-lcmd
usr/lib/llib-lcmd.ln
usr/lib/amd64/llib-lcmd.ln              i386
usr/lib/sparcv9/llib-lcmd.ln            sparc
# libdll
usr/lib/libdll.so
usr/lib/amd64/libdll.so                 i386
usr/lib/sparcv9/libdll.so               sparc
usr/lib/llib-ldll
usr/lib/llib-ldll.ln
usr/lib/amd64/llib-ldll.ln              i386
usr/lib/sparcv9/llib-ldll.ln            sparc
# libpp (a helper library needed by AST's msgcc)
usr/lib/libpp.so
usr/lib/llib-lpp
usr/lib/llib-lpp.ln
usr/lib/locale/C/LC_MESSAGES/libpp
# libshell
usr/lib/libshell.so
usr/lib/amd64/libshell.so               i386
usr/lib/sparcv9/libshell.so             sparc
usr/lib/llib-lshell
usr/lib/llib-lshell.ln
usr/lib/amd64/llib-lshell.ln            i386
usr/lib/sparcv9/llib-lshell.ln          sparc
# libsum
```



```
usr/lib/libsum.so
usr/lib/amd64/libsum.so                 i386
usr/lib/sparcv9/libsum.so               sparc
usr/lib/llib-lsum
usr/lib/llib-lsum.ln
usr/lib/amd64/llib-lsum.ln              i386
usr/lib/sparcv9/llib-lsum.ln            sparc
#
# This file is used in ON to build DSCP clients.  It is not for customers.
#
usr/include/libdscp.h           sparc
#
# These files are used by the iSCSI Target and the iSCSI Initiator
#
usr/include/sys/iscsi_protocol.h
usr/include/sys/iscsi_authclient.h
usr/include/sys/iscsi_authclientglue.h
#
# These files are used by the COMSTAR iSCSI target port provider
#
usr/include/sys/idm
usr/include/sys/iscsit/chap.h
usr/include/sys/iscsit/iscsi_if.h
usr/include/sys/iscsit/isns_protocol.h
usr/include/sys/iscsit/radius_packet.h
usr/include/sys/iscsit/radius_protocol.h
#
# libshare is private and the 64-bit sharemgr is not delivered.
#
usr/lib/libshare.so
usr/lib/amd64/libshare.so               i386
usr/lib/sparcv9/libshare.so             sparc
usr/lib/fs/autofs/libshare_autofs.so
usr/lib/fs/autofs/amd64/libshare_autofs.so              i386
usr/lib/fs/autofs/sparcv9/libshare_autofs.so            sparc
usr/lib/fs/nfs/libshare_nfs.so
usr/lib/fs/nfs/amd64/libshare_nfs.so            i386
usr/lib/fs/nfs/sparcv9/libshare_nfs.so          sparc
usr/lib/fs/nfs/test_svc_tp_create
usr/lib/fs/smb/libshare_smb.so
usr/lib/fs/smb/amd64/libshare_smb.so            i386
usr/lib/fs/smb/sparcv9/libshare_smb.so          sparc
usr/lib/fs/smbfs/libshare_smbfs.so
usr/lib/fs/smbfs/amd64/libshare_smbfs.so        i386
usr/lib/fs/smbfs/sparcv9/libshare_smbfs.so      sparc
usr/include/libshare_impl.h
usr/include/scfutil.h
#
# These files are installed in the proto area by the build of libpri for
# the benefit of the builds of FMA libldom, Zeus, picld plugins, and/or
# other libpri consumers. However, the libpri interfaces are private to
# Sun (Consolidation Private) and not intended for customer use. So these
# files (the symlink and the lint library) are excluded from packaging.
#
usr/lib/libpri.so               sparc
usr/lib/llib-lpri               sparc
usr/lib/llib-lpri.ln            sparc
usr/lib/sparcv9/libpri.so       sparc
usr/lib/sparcv9/llib-lpri.ln    sparc
#
# These files are installed in the proto area by the build of libds for
# the benefit of the builds of sun4v IO FMA and/or other libds
# consumers.  However, the libds interfaces are private to Sun
# (Consolidation Private) and not intended for customer use. So these
# files (the symlink and the lint library) are excluded from packaging.
#
```

```
usr/lib/libds.so                        sparc
usr/lib/sparcv9/libds.so                sparc
usr/lib/llib-lds                        sparc
usr/lib/llib-lds.ln                     sparc
usr/lib/sparcv9/llib-lds.ln             sparc
usr/platform/sun4v/include/sys/libds.h  sparc
usr/platform/sun4v/include/sys/vlds.h   sparc
#
# Private/Internal u8_textprep header file. Do not ship.
#
usr/include/sys/u8_textprep_data.h
#
# SQLite is private, used by SMF (svc.configd), idmapd and libsmb.
#
usr/include/sqlite-sys
lib/libsqlite-native.o
lib/libsqlite-sys.so
lib/llib-lsqlite-sys
lib/llib-lsqlite-sys.ln
#
# Private/Internal kiconv header files. Do not ship.
#
usr/include/sys/kiconv_big5_utf8.h
usr/include/sys/kiconv_cck_common.h
usr/include/sys/kiconv_cp950hkscs_utf8.h
usr/include/sys/kiconv_emea1.h
usr/include/sys/kiconv_emea2.h
usr/include/sys/kiconv_euckr_utf8.h
usr/include/sys/kiconv_euctw_utf8.h
usr/include/sys/kiconv_gb18030_utf8.h
usr/include/sys/kiconv_gb2312_utf8.h
usr/include/sys/kiconv_hkscs_utf8.h
usr/include/sys/kiconv_ja_jis_to_unicode.h
usr/include/sys/kiconv_ja_unicode_to_jis.h
usr/include/sys/kiconv_ja.h
usr/include/sys/kiconv_ko.h
usr/include/sys/kiconv_latin1.h
usr/include/sys/kiconv_sc.h
usr/include/sys/kiconv_tc.h
usr/include/sys/kiconv_uhc_utf8.h
usr/include/sys/kiconv_utf8_big5.h
usr/include/sys/kiconv_utf8_cp950hkscs.h
usr/include/sys/kiconv_utf8_euckr.h
usr/include/sys/kiconv_utf8_euctw.h
usr/include/sys/kiconv_utf8_gb18030.h
usr/include/sys/kiconv_utf8_gb2312.h
usr/include/sys/kiconv_utf8_hkscs.h
usr/include/sys/kiconv_utf8_uhc.h
#
# At this time, the directory and its contents
# are only useful on sun4u systems
#
etc/flash/postdeployment                        i386
#
# This header file is shared only between the power commands and
# ppm/srn modules # and should not be in any package
#
usr/include/sys/srn.h
#
# Private/Internal header files of smbsrv. Do not ship.
#
usr/include/smb
usr/include/smbsrv
#
# Private/Internal files for libfakekernel. Do not ship.
#
```



```
lib/amd64/libfakekernel.so                      i386
lib/amd64/llib-lfakekernel.ln                   i386
lib/sparcv9/libfakekernel.so                    sparc
lib/sparcv9/llib-lfakekernel.ln                 sparc
lib/libfakekernel.so
lib/llib-lfakekernel
lib/llib-lfakekernel.ln
usr/include/libfakekernel
usr/lib/libfakekernel.so
usr/lib/amd64/libfakekernel.so                  i386
usr/lib/sparcv9/libfakekernel.so                sparc
#
# Private/Internal libraries of smbsrv. Do not ship.
#
usr/lib/mdb/proc/libfksmbsrv.so
usr/lib/mdb/proc/amd64/libfksmbsrv.so           i386
usr/lib/mdb/proc/sparcv9/libfksmbsrv.so         sparc
usr/lib/reparse/llib-lreparse_smb
usr/lib/reparse/llib-lreparse_smb.ln
usr/lib/smbsrv/bind-helper
usr/lib/smbsrv/fksmbd
usr/lib/smbsrv/libfksmbsrv.so
usr/lib/smbsrv/libfksmbsrv.so.1
usr/lib/smbsrv/libmlsvc.so
usr/lib/smbsrv/libsmb.so
usr/lib/smbsrv/libsmbns.so
usr/lib/smbsrv/llib-lfksmbsrv
usr/lib/smbsrv/llib-lfksmbsrv.ln
usr/lib/smbsrv/llib-lmlsvc
usr/lib/smbsrv/llib-lmlsvc.ln
usr/lib/smbsrv/llib-lsmb
usr/lib/smbsrv/llib-lsmb.ln
usr/lib/smbsrv/llib-lsmbns
usr/lib/smbsrv/llib-lsmbns.ln
#
#
# Private/Internal 64-bit libraries of smbsrv. Do not ship.
#
usr/lib/smbsrv/amd64                            i386
usr/lib/smbsrv/sparcv9                          sparc

usr/lib/reparse/amd64/libreparse_smb.so         i386
usr/lib/reparse/amd64/libreparse_smb.so.1       i386
usr/lib/reparse/amd64/llib-lreparse_smb.ln      i386
usr/lib/reparse/sparcv9/libreparse_smb.so       sparc
usr/lib/reparse/sparcv9/libreparse_smb.so.1     sparc
usr/lib/reparse/sparcv9/llib-lreparse_smb.ln    sparc
#
# Private dirent, extended to include flags, for use by SMB server
#
usr/include/sys/extdirent.h
#
# Private header files for vscan service
#
usr/include/libvscan.h
usr/include/sys/vscan.h
#
# libvscan is private
#
usr/lib/vscan/llib-lvscan
usr/lib/vscan/llib-lvscan.ln
#
# i86hvm is not a full platform.  It is just a home for paravirtualized
# drivers.  There is no usr/ component to this sub-platform, but the
# directory is created in the proto area to keep other tools happy.
#
```

```
usr/platform/i86hvm                                     i386
#
# Private sdcard framework headers
#
usr/include/sys/sdcard
#
# libmlrpc is private (SMB client and server)
#
usr/include/libmlrpc
usr/lib/libmlrpc.so
usr/lib/amd64/libmlrpc.so               i386
usr/lib/amd64/libmlrpc.so.2             i386
usr/lib/sparcv9/libmlrpc.so             sparc
usr/lib/sparcv9/libmlrpc.so.2           sparc
usr/lib/llib-lmlrpc
usr/lib/llib-lmlrpc.ln
usr/lib/amd64/llib-lmlrpc.ln            i386
usr/lib/sparcv9/llib-lmlrpc.ln          sparc
#
# libsmbfs is private (SMB client and server)
#
usr/include/netsmb
usr/lib/libsmbfs.so
usr/lib/amd64/libsmbfs.so               i386
usr/lib/sparcv9/libsmbfs.so             sparc
usr/lib/llib-lsmbfs
usr/lib/llib-lsmbfs.ln
usr/lib/amd64/llib-lsmbfs.ln            i386
usr/lib/sparcv9/llib-lsmbfs.ln          sparc
#
# demo & test program for smbfs (private) ACL support
#
usr/lib/fs/smbfs/chacl
usr/lib/fs/smbfs/lsacl
opt/smbcl-tests
#
# FC related files
kernel/kmdb/amd64/fcip          i386
kernel/kmdb/sparcv9/fcip        sparc
kernel/kmdb/amd64/fcp           i386
kernel/kmdb/sparcv9/fcp         sparc
kernel/kmdb/amd64/fctl          i386
kernel/kmdb/sparcv9/fctl        sparc
kernel/kmdb/amd64/qlc           i386
kernel/kmdb/sparcv9/qlc         sparc
lib/llib-la5k                   sparc
lib/llib-la5k.ln                sparc
lib/sparcv9/llib-la5k.ln        sparc
lib/llib-lg_fc                  sparc
lib/llib-lg_fc.ln               sparc
lib/sparcv9/llib-lg_fc.ln       sparc
usr/include/a_state.h           sparc
usr/include/a5k.h               sparc
usr/include/exec.h              sparc
usr/include/g_scsi.h            sparc
usr/include/g_state.h           sparc
usr/include/gfc.h               sparc
usr/include/l_common.h          sparc
usr/include/l_error.h           sparc
usr/include/rom.h               sparc
usr/include/stgcom.h            sparc
usr/include/sys/fibre-channel
usr/lib/llib-lHBAAPI
usr/lib/llib-lHBAAPI.ln
usr/lib/amd64/llib-lHBAAPI.ln   i386
usr/lib/sparcv9/llib-lHBAAPI.ln sparc
```



```
#
# These files are used by the iSCSI initiator only.
# No reason to ship them.
#
usr/include/sys/scsi/adapters/iscsi_door.h
usr/include/sys/scsi/adapters/iscsi_if.h
#
# sbd ioctl hdr
#
usr/include/sys/stmf_sbd_ioctl.h
#
# proxy port provider interface
#
usr/include/sys/pppt_ic_if.h
usr/include/sys/pppt_ioctl.h
#
# proxy daemon lint library
#
usr/lib/llib-lstmfproxy
usr/lib/llib-lstmfproxy.ln
usr/lib/amd64/llib-lstmfproxy.ln        i386
usr/lib/sparcv9/llib-lstmfproxy.ln      sparc
#
# portable object file and dictionary used by libfmd_msg test
#
usr/lib/fm/dict/TEST.dict
usr/lib/locale/C/LC_MESSAGES/TEST.mo
usr/lib/locale/C/LC_MESSAGES/TEST.po
#
# Private idmap RPC protocol
#
usr/include/rpcsvc/idmap_prot.h
usr/include/rpcsvc/idmap_prot.x
#
# Private idmap directory API
#
usr/include/directory.h
#
# librstp is private for bridging
#
usr/include/stp_bpdu.h
usr/include/stp_in.h
usr/include/stp_vectors.h
usr/lib/librstp.so
usr/lib/llib-lrstp
usr/lib/llib-lrstp.ln
#
# Private nvfru API
#
usr/include/nvfru.h
#
# vrrp
#
usr/include/libvrrpadm.h
usr/lib/libvrrpadm.so
usr/lib/amd64/libvrrpadm.so             i386
usr/lib/sparcv9/libvrrpadm.so           sparc
usr/lib/llib-lvrrpadm
usr/lib/llib-lvrrpadm.ln
usr/lib/amd64/llib-lvrrpadm.ln          i386
usr/lib/sparcv9/llib-lvrrpadm.ln        sparc
#
# This is only used during the -t tools build
#
opt/onbld/bin/i386/mandoc       i386
opt/onbld/bin/sparc/mandoc      sparc
```

```
opt/onbld/bin/i386/makesoftcore i386
opt/onbld/bin/i386/vtfontcvt    i386
opt/onbld/bin/sparc/makesoftcore        sparc
opt/onbld/bin/sparc/vtfontcvt   sparc

#
# Private libdwarf
#
opt/onbld/lib/i386/libdwarf.so  i386
opt/onbld/lib/sparc/libdwarf.so sparc

#
# Private socket filter API
#
usr/include/sys/sockfilter.h
#
# We don't actually validate license action payloads, and the license
# staging area is provided as a separate basedir for package
# publication.  The net result is that everything therein should be
# ignored for packaging validation.
#
licenses
#
# Libbe is private
#
usr/include/libbe_priv.h
#
# ipmi is at present only useful on i386, but for historical reasons is
# delivered on SPARC and used by the build.
#
usr/include/sys/ipmi.h  sparc

#
# libsaveargs is private
#
usr/include/saveargs.h                  i386
usr/lib/amd64/libsaveargs.so            i386
usr/lib/amd64/libstandsaveargs.so       i386
usr/lib/amd64/llib-lsaveargs.ln         i386

#
# libpcidb is private
#
usr/include/pcidb.h
usr/lib/amd64/libpcidb.so               i386
usr/lib/amd64/llib-lpcidb.ln            i386
usr/lib/sparcv9/libpcidb.so             sparc
usr/lib/sparcv9/llib-lpcidb.ln          sparc
usr/lib/libpcidb.so
usr/lib/llib-lpcidb
usr/lib/llib-lpcidb.ln

#
# private nvme header file
#
usr/include/sys/nvme.h

#
# debugging program for libadutils
#
usr/bin/test-getdc
#
# libficl-sys is private
#
usr/include/ficllocal.h
usr/lib/amd64/llib-lficl-sys.ln         i386
```



```
usr/lib/amd64/libficl-sys.so            i386
usr/lib/sparcv9/llib-lficl-sys.ln       sparc
usr/lib/sparcv9/libficl-sys.so          sparc
usr/lib/llib-lficl-sys
usr/lib/llib-lficl-sys.ln
usr/lib/libficl-sys.so

#
# libsff is private
#
usr/include/libsff.h
usr/lib/amd64/libsff.so                 i386
usr/lib/amd64/llib-lsff.ln              i386
usr/lib/sparcv9/libsff.so               sparc
usr/lib/sparcv9/llib-lsff.ln            sparc
usr/lib/libsff.so
usr/lib/llib-lsff
usr/lib/llib-lsff.ln

#
# libcustr is private
#
usr/include/libcustr.h
lib/amd64/libcustr.so           i386
lib/amd64/llib-lcustr.ln        i386
lib/sparcv9/libcustr.so         sparc
lib/sparcv9/llib-lcustr.ln      sparc
lib/libcustr.so
lib/llib-lcustr
lib/llib-lcustr.ln

#
# smatch is delivered and used only with the source tree
#
opt/onbld/bin/i386/smatch       i386
opt/onbld/share/smatch
```

```
**********************************************************
     504 Fri Dec 21 14:59:54 2018
new/exception_lists/wscheck
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 #
   2 # This file and its contents are supplied under the terms of the
   3 # Common Development and Distribution License ("CDDL"), version 1.0.
   4 # You may only use this file in accordance with the terms of version
   5 # 1.0 of the CDDL.
   6 #
   7 # A full copy of the text of the CDDL should have accompanied this
   8 # source.  A copy of the CDDL is also available via the Internet at
   9 # http://www.illumos.org/license/CDDL.
  10 #
  11 # Copyright 2018 Joyent, Inc.
  12 #
  13 syntax: glob

  15 usr/src/uts/common/io/qede/*
  16 usr/src/tools/smatch/src/*
```

```
**********************************************************
   36643 Fri Dec 21 14:59:54 2018
new/usr/src/Makefile.master
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1989, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2012 by Delphix. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
# Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2015 Gary Mills
# Copyright 2015 Igor Kozhukhov <ikozhukhov@gmail.com>
# Copyright 2016 Toomas Soome <tsoome@me.com>
# Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
# Copyright (c) 2018, Joyent, Inc.
#

#
# Makefile.master, global definitions for system source
#
ROOT=           /proto

#
# Adjunct root, containing an additional proto area to be used for headers
# and libraries.
#
ADJUNCT_PROTO=

#
# Adjunct for building things that run on the build machine.
#
NATIVE_ADJUNCT= /usr

#
# RELEASE_BUILD should be cleared for final release builds.
# NOT_RELEASE_BUILD is exactly what the name implies.
#
# __GNUC toggles the building of ON components using gcc and related tools.
# Normally set to '#', set it to '' to do gcc build.
#
# The declaration POUND_SIGN is always '#'. This is needed to get around the
# make feature that '#' is always a comment delimiter, even when escaped or
# quoted. We use this macro expansion method to get POUND_SIGN rather than
# always breaking out a shell because the general case can cause a noticable
```



```
# slowdown in build times when so many Makefiles include Makefile.master.
#
# While the majority of users are expected to override the setting below
# with an env file (via nightly or bldenv), if you aren't building that way
# (ie, you're using "ws" or some other bootstrapping method) then you need
# this definition in order to avoid the subshell invocation mentioned above.
#

PRE_POUND=                              pre\#
POUND_SIGN=                             $(PRE_POUND:pre\%=%)

NOT_RELEASE_BUILD=
RELEASE_BUILD=                          $(POUND_SIGN)
$(RELEASE_BUILD)NOT_RELEASE_BUILD=      $(POUND_SIGN)
PATCH_BUILD=                            $(POUND_SIGN)

# SPARC_BLD is '#' for an Intel build.
# INTEL_BLD is '#' for a Sparc build.
SPARC_BLD_1=    $(MACH:i386=$(POUND_SIGN))
SPARC_BLD=      $(SPARC_BLD_1:sparc=)
INTEL_BLD_1=    $(MACH:sparc=$(POUND_SIGN))
INTEL_BLD=      $(INTEL_BLD_1:i386=)

# The variables below control the compilers used during the build.
# There are a number of permutations.
#
# __GNUC and __SUNC control (and indicate) the primary compiler.  Whichever
# one is not POUND_SIGN is the primary, with the other as the shadow.  They
# may also be used to control entirely compiler-specific Makefile assignments.
# __GNUC and GCC are the default.
#
# __GNUC64 indicates that the 64bit build should use the GNU C compiler.
# There is no Sun C analogue.
#
# The following version-specific options are operative regardless of which
# compiler is primary, and control the versions of the given compilers to be
# used.  They also allow compiler-version specific Makefile fragments.
#

__SUNC=                 $(POUND_SIGN)
$(__SUNC)__GNUC=        $(POUND_SIGN)
__GNUC64=               $(__GNUC)

# Allow build-time "configuration" to enable or disable some things.
# The default is POUND_SIGN, meaning "not enabled". If the environment
# passes in an override like ENABLE_SMB_PRINTING= (empty) that will
# uncomment things in the lower Makefiles to enable the feature.
ENABLE_SMB_PRINTING=    $(POUND_SIGN)

# CLOSED is the root of the tree that contains source which isn't released
# as open source
CLOSED=         $(SRC)/../closed

# BUILD_TOOLS is the root of all tools including compilers.
# ONBLD_TOOLS is the root of all the tools that are part of SUNWonbld.

BUILD_TOOLS=            /ws/onnv-tools
ONBLD_TOOLS=            $(BUILD_TOOLS)/onbld

# define runtime JAVA_HOME, primarily for cmd/pools/poold
JAVA_HOME=      /usr/java
# define buildtime JAVA_ROOT
JAVA_ROOT=      /usr/java
# Build uses java7 by default.  Pass one the variables below set to empty
# string in the environment to override.
BLD_JAVA_6=     $(POUND_SIGN)
```

```
127 BLD_JAVA_8=     $(POUND_SIGN)

129 GNUC_ROOT=      /opt/gcc/4.4.4
130 GCCLIBDIR=      $(GNUC_ROOT)/lib
131 GCCLIBDIR64=    $(GNUC_ROOT)/lib/$(MACH64)

133 DOCBOOK_XSL_ROOT=       /usr/share/sgml/docbook/xsl-stylesheets

135 RPCGEN=         /usr/bin/rpcgen
136 STABS=          $(ONBLD_TOOLS)/bin/$(MACH)/stabs
137 ELFEXTRACT=     $(ONBLD_TOOLS)/bin/$(MACH)/elfextract
138 MBH_PATCH=      $(ONBLD_TOOLS)/bin/$(MACH)/mbh_patch
139 BTXLD=          $(ONBLD_TOOLS)/bin/$(MACH)/btxld
140 VTFONTCVT=      $(ONBLD_TOOLS)/bin/$(MACH)/vtfontcvt
141 # echo(1) and true(1) are specified without absolute paths, so that the shell
142 # spawned by make(1) may use the built-in versions.  This is minimally
143 # problematic, as the shell spawned by make(1) is known and under control, the
144 # only risk being if the shell falls back to $PATH.
145 #
146 # We specifically want an echo(1) that does interpolation of escape sequences,
147 # which ksh93, /bin/sh, and bash will all provide.
148 ECHO=           echo
149 TRUE=           true
150 INS=            $(ONBLD_TOOLS)/bin/$(MACH)/install
151 SYMLINK=        /usr/bin/ln -s
152 LN=             /usr/bin/ln
153 MKDIR=          /usr/bin/mkdir
154 CHMOD=          /usr/bin/chmod
155 MV=             /usr/bin/mv -f
156 RM=             /usr/bin/rm -f
157 CUT=            /usr/bin/cut
158 NM=             /usr/ccs/bin/nm
159 DIFF=           /usr/bin/diff
160 GREP=           /usr/bin/grep
161 EGREP=          /usr/bin/egrep
162 ELFWRAP=        /usr/bin/elfwrap
163 KSH93=          /usr/bin/ksh93
164 SED=            /usr/bin/sed
165 AWK=            /usr/bin/nawk
166 CP=             /usr/bin/cp -f
167 MCS=            /usr/ccs/bin/mcs
168 CAT=            /usr/bin/cat
169 ELFDUMP=        /usr/ccs/bin/elfdump
170 M4=             /usr/bin/m4
171 STRIP=          /usr/ccs/bin/strip
172 LEX=            /usr/ccs/bin/lex
173 FLEX=           /usr/bin/flex
174 YACC=           /usr/ccs/bin/yacc
175 CPP=            /usr/lib/cpp
176 ANSI_CPP=       $(GNUC_ROOT)/bin/cpp
177 JAVAC=          $(JAVA_ROOT)/bin/javac
178 JAVAH=          $(JAVA_ROOT)/bin/javah
179 JAVADOC=        $(JAVA_ROOT)/bin/javadoc
180 RMIC=           $(JAVA_ROOT)/bin/rmic
181 JAR=            $(JAVA_ROOT)/bin/jar
182 CTFCONVERT=     $(ONBLD_TOOLS)/bin/$(MACH)/ctfconvert
183 CTFMERGE=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfmerge
184 CTFSTABS=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstabs
185 CTFSTRIP=       $(ONBLD_TOOLS)/bin/$(MACH)/ctfstrip
186 NDRGEN=         $(ONBLD_TOOLS)/bin/$(MACH)/ndrgen
187 GENOFFSETS=     $(ONBLD_TOOLS)/bin/genoffsets
188 XREF=           $(ONBLD_TOOLS)/bin/xref
189 FIND=           /usr/bin/find
190 PERL=           /usr/bin/perl
191 PERL_VERSION=   5.10.0
192 PERL_PKGVERS=   -510
```



```
193 PERL_ARCH =             i86pc-solaris-64int
194 $(SPARC_BLD)PERL_ARCH = sun4-solaris-64int
195 PYTHON_VERSION= 2.7
196 PYTHON_PKGVERS= -27
197 PYTHON_SUFFIX=
198 PYTHON=          /usr/bin/python$(PYTHON_VERSION)
199 PYTHON3_VERSION=        3.5
200 PYTHON3_PKGVERS=        -35
201 PYTHON3_SUFFIX=         m
202 PYTHON3=               /usr/bin/python$(PYTHON3_VERSION)
203 SORT=           /usr/bin/sort
204 TR=             /usr/bin/tr
205 TOUCH=          /usr/bin/touch
206 WC=             /usr/bin/wc
207 XARGS=          /usr/bin/xargs
208 ELFEDIT=        /usr/bin/elfedit
209 DTRACE=         /usr/sbin/dtrace -xnolibs
210 UNIQ=           /usr/bin/uniq
211 TAR=            /usr/bin/tar
212 ASTBINDIR=      /usr/ast/bin
213 MSGCC=          $(ASTBINDIR)/msgcc
214 MSGFMT=         /usr/bin/msgfmt -s
215 LCDEF=          $(ONBLD_TOOLS)/bin/$(MACH)/localedef
216 TIC=            $(ONBLD_TOOLS)/bin/$(MACH)/tic
217 ZIC=            $(ONBLD_TOOLS)/bin/$(MACH)/zic
218 OPENSSL=        /usr/bin/openssl

220 FILEMODE=       644
221 DIRMODE=        755

223 # Declare that nothing should be built in parallel.
224 # Individual Makefiles can use the .PARALLEL target to declare otherwise.
225 .NO_PARALLEL:

227 # For stylistic checks
228 #
229 # Note that the X and C checks are not used at this time and may need
230 # modification when they are actually used.
231 #
232 CSTYLE=         $(ONBLD_TOOLS)/bin/cstyle
233 CSTYLE_TAIL=
234 HDRCHK=         $(ONBLD_TOOLS)/bin/hdrchk
235 HDRCHK_TAIL=
236 JSTYLE=         $(ONBLD_TOOLS)/bin/jstyle

238 DOT_H_CHECK=    \
239         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL); \
240         $(HDRCHK) $< $(HDRCHK_TAIL)

242 DOT_X_CHECK=    \
243         @$(ECHO) "checking $<"; $(RPCGEN) -C -h $< | $(CSTYLE) $(CSTYLE_TAIL); \
244         $(RPCGEN) -C -h $< | $(HDRCHK) $< $(HDRCHK_TAIL)

246 DOT_C_CHECK=    \
247         @$(ECHO) "checking $<"; $(CSTYLE) $< $(CSTYLE_TAIL)

249 MANIFEST_CHECK= \
250         @$(ECHO) "checking $<"; \
251         SVCCFG_DTD=$(SRC)/cmd/svc/dtd/service_bundle.dtd.1 \
252         SVCCFG_REPOSITORY=$(SRC)/cmd/svc/seed/global.db \
253         SVCCFG_CONFIGD_PATH=$(SRC)/cmd/svc/configd/svc.configd-native \
254         $(SRC)/cmd/svc/svccfg/svccfg-native validate $<

256 INS.file=       $(RM) $@; $(INS) -s -m $(FILEMODE) -f $(@D) $<
257 INS.dir=        $(INS) -s -d -m $(DIRMODE) $@
258 # installs and renames at once
```

```
259 #
260 INS.rename=     $(INS.file); $(MV) $(@D)/$(<F) $@

262 # install a link
263 INSLINKTARGET=  $<
264 INS.link=       $(RM) $@; $(LN) $(INSLINKTARGET) $@
265 INS.symlink=    $(RM) $@; $(SYMLINK) $(INSLINKTARGET) $@

267 #
268 # Python bakes the mtime of the .py file into the compiled .pyc and
269 # rebuilds if the baked-in mtime != the mtime of the source file
270 # (rather than only if it's less than), thus when installing python
271 # files we must make certain to not adjust the mtime of the source
272 # (.py) file.
273 #
274 INS.pyfile=     $(RM) $@; $(SED) -e "1s:^\#!@PYTHON@:\#!$(PYTHON):" < $< > $@; $

276 # MACH must be set in the shell environment per uname -p on the build host
277 # More specific architecture variables should be set in lower makefiles.
278 #
279 # MACH64 is derived from MACH, and BUILD64 is set to '#' for
280 # architectures on which we do not build 64-bit versions.
281 # (There are no such architectures at the moment.)
282 #
283 # Set BUILD64=# in the environment to disable 64-bit amd64
284 # builds on i386 machines.

286 MACH64_1=       $(MACH:sparc=sparcv9)
287 MACH64=         $(MACH64_1:i386=amd64)

289 MACH32_1=       $(MACH:sparc=sparcv7)
290 MACH32=         $(MACH32_1:i386=i86)

292 sparc_BUILD64=
293 i386_BUILD64=
294 BUILD64=        $($(MACH)_BUILD64)

296 #
297 # C compiler mode. Future compilers may change the default on us,
298 # so force extended ANSI mode globally. Lower level makefiles can
299 # override this by setting CCMODE.
300 #
301 CCMODE=                 -Xa
302 CCMODE64=               -Xa

304 #
305 # C compiler verbose mode. This is so we can enable it globally,
306 # but turn it off in the lower level makefiles of things we cannot
307 # (or aren't going to) fix.
308 #
309 CCVERBOSE=              -v

311 # set this to the secret flag "-Wc,-Qiselect-v9abiwarn=1" to get warnings
312 # from the compiler about places the -xarch=v9 may differ from -xarch=v9c.
313 V9ABIWARN=

315 # set this to the secret flag "-Wc,-Qiselect-regsym=0" to disable register
316 # symbols (used to detect conflicts between objects that use global registers)
317 # we disable this now for safety, and because genunix doesn't link with
318 # this feature (the v9 default) enabled.
319 #
320 # REGSYM is separate since the C++ driver syntax is different.
321 CCREGSYM=               -Wc,-Qiselect-regsym=0
322 CCCREGSYM=              -Qoption cg -Qiselect-regsym=0

324 # Prevent the removal of static symbols by the SPARC code generator (cg).
```



```
325 # The x86 code generator (ube) does not remove such symbols and as such
326 # using this workaround is not applicable for x86.
327 #
328 CCSTATICSYM=            -Wc,-Qassembler-ounrefsym=0
329 #
330 # generate 32-bit addresses in the v9 kernel. Saves memory.
331 CCABS32=                -Wc,-xcode=abs32
332 #
333 # generate v9 code which tolerates callers using the v7 ABI, for the sake of
334 # system calls.
335 CC32BITCALLERS=         -_gcc=-massume-32bit-callers

337 # GCC, especially, is increasingly beginning to auto-inline functions and
338 # sadly does so separately not under the general -fno-inline-functions
339 # Additionally, we wish to prevent optimisations which cause GCC to clone
340 # functions -- in particular, these may cause unhelpful symbols to be
341 # emitted instead of function names
342 CCNOAUTOINLINE= \
343         -_gcc=-fno-inline-small-functions \
344         -_gcc=-fno-inline-functions-called-once \
345         -_gcc=-fno-ipa-cp \
346         -_gcc7=-fno-ipa-icf \
347         -_gcc8=-fno-ipa-icf \
348         -_gcc7=-fno-clone-functions \
349         -_gcc8=-fno-clone-functions

351 # GCC may put functions in different named sub-sections of .text based on
352 # their presumed calling frequency.  At least in the kernel, where we actually
353 # deliver relocatable objects, we don't want this to happen.
354 #
355 # Since at present we don't benefit from this even in userland, we disable it gl
356 # but the application of this may move into usr/src/uts/ in future.
357 CCNOREORDER=    \
358         -_gcc7=-fno-reorder-functions \
359         -_gcc8=-fno-reorder-functions

361 # One optimization the compiler might perform is to turn this:
362 #       #pragma weak foo
363 #       extern int foo;
364 #       if (&foo)
365 #               foo = 5;
366 # into
367 #       foo = 5;
368 # Since we do some of this (foo might be referenced in common kernel code
369 # but provided only for some cpu modules or platforms), we disable this
370 # optimization.
371 #
372 sparc_CCUNBOUND = -Wd,-xsafe=unboundsym
373 i386_CCUNBOUND  =
374 CCUNBOUND       = $($(MACH)_CCUNBOUND)

376 #
377 # compiler '-xarch' flag. This is here to centralize it and make it
378 # overridable for testing.
379 sparc_XARCH=    -m32
380 sparcv9_XARCH=  -m64
381 i386_XARCH=     -m32
382 amd64_XARCH=    -m64 -Ui386 -U__i386

384 # assembler '-xarch' flag.  Different from compiler '-xarch' flag.
385 sparc_AS_XARCH=         -xarch=v8plus
386 sparcv9_AS_XARCH=       -xarch=v9
387 i386_AS_XARCH=
388 amd64_AS_XARCH=         -xarch=amd64 -P -Ui386 -U__i386

390 #
```

```
391 # These flags define what we need to be 'standalone' i.e. -not- part
392 # of the rather more cosy userland environment.  This basically means
393 # the kernel.
394 #
395 # XX64  future versions of gcc will make -mcmodel=kernel imply -mno-red-zone
396 #
397 sparc_STAND_FLAGS=      -_gcc=-ffreestanding
398 sparcv9_STAND_FLAGS=    -_gcc=-ffreestanding
399 # Disabling MMX also disables 3DNow, disabling SSE also disables all later
400 # additions to SSE (SSE2, AVX ,etc.)
401 NO_SIMD=                -_gcc=-mno-mmx -_gcc=-mno-sse
402 i386_STAND_FLAGS=       -_gcc=-ffreestanding $(NO_SIMD)
403 amd64_STAND_FLAGS=      -xmodel=kernel $(NO_SIMD)

405 SAVEARGS=               -Wu,-save_args
406 amd64_STAND_FLAGS       += $(SAVEARGS)

408 STAND_FLAGS_32 = $($(MACH)_STAND_FLAGS)
409 STAND_FLAGS_64 = $($(MACH64)_STAND_FLAGS)

411 #
412 # disable the incremental linker
413 ILDOFF=                 -xildoff
414 #
415 XFFLAG=                 -xF=%all
416 XESS=                   -xs
417 XSTRCONST=              -xstrconst

419 #
420 # turn warnings into errors (C)
421 CERRWARN = -errtags=yes -errwarn=%all
422 CERRWARN += -erroff=E_EMPTY_TRANSLATION_UNIT
423 CERRWARN += -erroff=E_STATEMENT_NOT_REACHED

425 CERRWARN += -_gcc=-Wno-missing-braces
426 CERRWARN += -_gcc=-Wno-sign-compare
427 CERRWARN += -_gcc=-Wno-unknown-pragmas
428 CERRWARN += -_gcc=-Wno-unused-parameter
429 CERRWARN += -_gcc=-Wno-missing-field-initializers

431 # Unfortunately, this option can misfire very easily and unfixably.
432 CERRWARN +=     -_gcc=-Wno-array-bounds

434 # DEBUG v. -nd make for frequent unused variables, empty conditions, etc. in
435 # -nd builds
436 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused
437 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-empty-body

439 CERRWARN += -_smatch=-p=illumos_user
440 include $(SRC)/Makefile.smatch

442 #
443 # turn warnings into errors (C++)
444 CCERRWARN=              -xwe

446 # C standard.  Keep Studio flags until we get rid of lint.
447 CSTD_GNU89=     -xc99=%none
448 CSTD_GNU99=     -xc99=%all
449 CSTD=           $(CSTD_GNU89)
450 C99LMODE=       $(CSTD:-xc99%=-Xc99%)

452 # In most places, assignments to these macros should be appended with +=
453 # (CPPFLAGS.first allows values to be prepended to CPPFLAGS).
454 sparc_CFLAGS=   $(sparc_XARCH) $(CCSTATICSYM)
455 sparcv9_CFLAGS= $(sparcv9_XARCH) -dalign $(CCVERBOSE) $(V9ABIWARN) $(CCREGSYM) \
456                 $(CCSTATICSYM)
```



```
457 i386_CFLAGS=    $(i386_XARCH)
458 amd64_CFLAGS=   $(amd64_XARCH)

460 sparc_ASFLAGS=  $(sparc_AS_XARCH)
461 sparcv9_ASFLAGS=$(sparcv9_AS_XARCH)
462 i386_ASFLAGS=   $(i386_AS_XARCH)
463 amd64_ASFLAGS=  $(amd64_AS_XARCH)

465 #
466 sparc_COPTFLAG=         -xO3
467 sparcv9_COPTFLAG=       -xO3
468 i386_COPTFLAG=          -O
469 amd64_COPTFLAG=         -xO3

471 COPTFLAG= $($(MACH)_COPTFLAG)
472 COPTFLAG64= $($(MACH64)_COPTFLAG)

474 # When -g is used, the compiler globalizes static objects
475 # (gives them a unique prefix). Disable that.
476 CNOGLOBAL= -W0,-noglobal

478 # Direct the Sun Studio compiler to use a static globalization prefix based on t
479 # name of the module rather than something unique. Otherwise, objects
480 # will not build deterministically, as subsequent compilations of identical
481 # source will yeild objects that always look different.
482 #
483 # In the same spirit, this will also remove the date from the N_OPT stab.
484 CGLOBALSTATIC= -W0,-xglobalstatic

486 # Sometimes we want all symbols and types in debugging information even
487 # if they aren't used.
488 CALLSYMS=       -W0,-xdbggen=no%usedonly

490 #
491 # Default debug format for Sun Studio 11 is dwarf, so force it to
492 # generate stabs.
493 #
494 DEBUGFORMAT=    -xdebugformat=stabs

496 #
497 # Flags used to build in debug mode for ctf generation.  Bugs in the Devpro
498 # compilers currently prevent us from building with cc-emitted DWARF.
499 #
500 CTF_FLAGS_sparc = -g -Wc,-Qiselect-T1 $(CSTD) $(CNOGLOBAL) $(CDWARFSTR)
501 CTF_FLAGS_i386  = -g $(CSTD) $(CNOGLOBAL) $(CDWARFSTR)

503 CTF_FLAGS_sparcv9       = $(CTF_FLAGS_sparc)
504 CTF_FLAGS_amd64         = $(CTF_FLAGS_i386)

506 # Sun Studio produces broken userland code when saving arguments.
507 $(__GNUC)CTF_FLAGS_amd64 += $(SAVEARGS)

509 CTF_FLAGS_32    = $(CTF_FLAGS_$(MACH)) $(DEBUGFORMAT)
510 CTF_FLAGS_64    = $(CTF_FLAGS_$(MACH64)) $(DEBUGFORMAT)
511 CTF_FLAGS       = $(CTF_FLAGS_32)

513 #
514 # Flags used with genoffsets
515 #
516 GOFLAGS = $(CALLSYMS) $(CDWARFSTR)

518 OFFSETS_CREATE = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
519         $(CW) --noecho $(CW_CC_COMPILERS) -- $(GOFLAGS) $(CFLAGS) $(CPPFLAGS)

521 OFFSETS_CREATE64 = $(GENOFFSETS) -s $(CTFSTABS) -r $(CTFCONVERT) \
522         $(CW) --noecho $(CW_CC_COMPILERS) -- $(GOFLAGS) $(CFLAGS64) $(CPPFLAGS)
```

```
524 #
525 # tradeoff time for space (smaller is better)
526 #
527 sparc_SPACEFLAG         = -xspace -W0,-Lt
528 sparcv9_SPACEFLAG       = -xspace -W0,-Lt
529 i386_SPACEFLAG          = -xspace
530 amd64_SPACEFLAG         =

532 SPACEFLAG               = $($(MACH)_SPACEFLAG)
533 SPACEFLAG64             = $($(MACH64)_SPACEFLAG)

535 #
536 # The Sun Studio 11 compiler has changed the behaviour of integer
537 # wrap arounds and so a flag is needed to use the legacy behaviour
538 # (without this flag panics/hangs could be exposed within the source).
539 #
540 sparc_IROPTFLAG         = -W2,-xwrap_int
541 sparcv9_IROPTFLAG       = -W2,-xwrap_int
542 i386_IROPTFLAG          =
543 amd64_IROPTFLAG         =

545 IROPTFLAG               = $($(MACH)_IROPTFLAG)
546 IROPTFLAG64             = $($(MACH64)_IROPTFLAG)

548 sparc_XREGSFLAG         = -xregs=no%appl
549 sparcv9_XREGSFLAG       = -xregs=no%appl
550 i386_XREGSFLAG          =
551 amd64_XREGSFLAG         =

553 XREGSFLAG               = $($(MACH)_XREGSFLAG)
554 XREGSFLAG64             = $($(MACH64)_XREGSFLAG)

556 # dmake SOURCEDEBUG=yes ... enables source-level debugging information, and
557 # avoids stripping it.
558 SOURCEDEBUG     = $(POUND_SIGN)
559 SRCDBGBLD       = $(SOURCEDEBUG:yes=)

561 #
562 # These variables are intended ONLY for use by developers to safely pass extra
563 # flags to the compilers without unintentionally overriding Makefile-set
564 # flags.  They should NEVER be set to any value in a Makefile.
565 #
566 # They come last in the associated FLAGS variable such that they can
567 # explicitly override things if necessary, there are gaps in this, but it's
568 # the best we can manage.
569 #
570 CUSERFLAGS              =
571 CUSERFLAGS64            = $(CUSERFLAGS)
572 CCUSERFLAGS             =
573 CCUSERFLAGS64           = $(CCUSERFLAGS)

575 CSOURCEDEBUGFLAGS       =
576 CCSOURCEDEBUGFLAGS      =
577 $(SRCDBGBLD)CSOURCEDEBUGFLAGS   = -g -xs
578 $(SRCDBGBLD)CCSOURCEDEBUGFLAGS  = -g -xs

580 CFLAGS=         $(COPTFLAG) $($(MACH)_CFLAGS) $(SPACEFLAG) $(CCMODE) \
581                 $(ILDOFF) $(CERRWARN) $(CSTD) $(CCUNBOUND) $(IROPTFLAG) \
582                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CCNOREORDER) \
583                 $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)
584 CFLAGS64=       $(COPTFLAG64) $($(MACH64)_CFLAGS) $(SPACEFLAG64) $(CCMODE64) \
585                 $(ILDOFF) $(CERRWARN) $(CSTD) $(CCUNBOUND) $(IROPTFLAG64) \
586                 $(CGLOBALSTATIC) $(CCNOAUTOINLINE) $(CCNOREORDER) \
587                 $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS64)
588 #
```



```
589 # Flags that are used to build parts of the code that are subsequently
590 # run on the build machine (also known as the NATIVE_BUILD).
591 #
592 NATIVE_CFLAGS=  $(COPTFLAG) $($(NATIVE_MACH)_CFLAGS) $(CCMODE) \
593                 $(ILDOFF) $(CERRWARN) $(CSTD) $($(NATIVE_MACH)_CCUNBOUND) \
594                 $(IROPTFLAG) $(CGLOBALSTATIC) $(CCNOAUTOINLINE) \
595                 $(CCNOREORDER) $(CSOURCEDEBUGFLAGS) $(CUSERFLAGS)

597 DTEXTDOM=-DTEXT_DOMAIN=\"$(TEXT_DOMAIN)\"       # For messaging.
598 DTS_ERRNO=-D_TS_ERRNO
599 CPPFLAGS.first= # Please keep empty.  Only lower makefiles should set this.
600 CPPFLAGS.master=$(DTEXTDOM) $(DTS_ERRNO) \
601         $(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) $(ENVCPPFLAGS4) \
602         $(ADJUNCT_PROTO:%=-I%/usr/include)
603 CPPFLAGS.native=$(ENVCPPFLAGS1) $(ENVCPPFLAGS2) $(ENVCPPFLAGS3) \
604                 $(ENVCPPFLAGS4) -I$(NATIVE_ADJUNCT)/include
605 CPPFLAGS=       $(CPPFLAGS.first) $(CPPFLAGS.master)
606 AS_CPPFLAGS=    $(CPPFLAGS.first) $(CPPFLAGS.master)
607 JAVAFLAGS=      -source 1.6 -target 1.6 -Xlint:deprecation,-options

609 #
610 # For source message catalogue
611 #
612 .SUFFIXES: $(SUFFIXES) .i .po
613 MSGROOT= $(ROOT)/catalog
614 MSGDOMAIN= $(MSGROOT)/$(TEXT_DOMAIN)
615 MSGDOMAINPOFILE = $(MSGDOMAIN)/$(POFILE)
616 DCMSGDOMAIN= $(MSGROOT)/LC_TIME/$(TEXT_DOMAIN)
617 DCMSGDOMAINPOFILE = $(DCMSGDOMAIN)/$(DCFILE:.dc=.po)

619 CLOBBERFILES += $(POFILE) $(POFILES)
620 COMPILE.cpp= $(CC) -E -C $(CFLAGS) $(CPPFLAGS)
621 XGETTEXT= /usr/bin/xgettext
622 XGETFLAGS= -c TRANSLATION_NOTE
623 GNUXGETTEXT= /usr/gnu/bin/xgettext
624 GNUXGETFLAGS= --add-comments=TRANSLATION_NOTE --keyword=_ \
625         --strict --no-location --omit-header
626 BUILD.po= $(XGETTEXT) $(XGETFLAGS) -d $(<F) $<.i ;\
627         $(RM)    $@ ;\
628         $(SED) "/^domain/d" < $(<F).po > $@ ;\
629         $(RM) $(<F).po $<.i

631 #
632 # This is overwritten by local Makefile when PROG is a list.
633 #
634 POFILE= $(PROG).po

636 sparc_CCFLAGS=          -cg92 -compat=4 \
637                         -Qoption ccfe -messages=no%anachronism \
638                         $(CCERRWARN)
639 sparcv9_CCFLAGS=        $(sparcv9_XARCH) -dalign -compat=5 \
640                         -Qoption ccfe -messages=no%anachronism \
641                         -Qoption ccfe -features=no%conststrings \
642                         $(CCCREGSYM) \
643                         $(CCERRWARN)
644 i386_CCFLAGS=           -compat=4 \
645                         -Qoption ccfe -messages=no%anachronism \
646                         -Qoption ccfe -features=no%conststrings \
647                         $(CCERRWARN)
648 amd64_CCFLAGS=          $(amd64_XARCH) -compat=5 \
649                         -Qoption ccfe -messages=no%anachronism \
650                         -Qoption ccfe -features=no%conststrings \
651                         $(CCERRWARN)

653 sparc_CCOPTFLAG=        -O
654 sparcv9_CCOPTFLAG=      -O
```

```
655 i386_CCOPTFLAG=         -O
656 amd64_CCOPTFLAG=        -O

658 CCOPTFLAG=      $($(MACH)_CCOPTFLAG)
659 CCOPTFLAG64=    $($(MACH64)_CCOPTFLAG)
660 CCFLAGS=        $(CCOPTFLAG) $($(MACH)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
661                 $(CCUSERFLAGS)
662 CCFLAGS64=      $(CCOPTFLAG64) $($(MACH64)_CCFLAGS) $(CCSOURCEDEBUGFLAGS) \
663                 $(CCUSERFLAGS64)

665 #
666 #
667 #
668 ELFWRAP_FLAGS   =
669 ELFWRAP_FLAGS64 =       -64

671 #
672 # Various mapfiles that are used throughout the build, and delivered to
673 # /usr/lib/ld.
674 #
675 MAPFILE.NED_i386 =      $(SRC)/common/mapfiles/common/map.noexdata
676 MAPFILE.NED_sparc =
677 MAPFILE.NED =           $(MAPFILE.NED_$(MACH))
678 MAPFILE.PGA =           $(SRC)/common/mapfiles/common/map.pagealign
679 MAPFILE.NES =           $(SRC)/common/mapfiles/common/map.noexstk
680 MAPFILE.FLT =           $(SRC)/common/mapfiles/common/map.filter
681 MAPFILE.LEX =           $(SRC)/common/mapfiles/common/map.lex.yy

683 #
684 # Generated mapfiles that are compiler specific, and used throughout the
685 # build.  These mapfiles are not delivered in /usr/lib/ld.
686 #
687 MAPFILE.NGB_sparc=      $(SRC)/common/mapfiles/gen/sparc_cc_map.noexeglobs
688 $(__GNUC64)MAPFILE.NGB_sparc= \
689                         $(SRC)/common/mapfiles/gen/sparc_gcc_map.noexeglobs
690 MAPFILE.NGB_sparcv9=    $(SRC)/common/mapfiles/gen/sparcv9_cc_map.noexeglobs
691 $(__GNUC64)MAPFILE.NGB_sparcv9= \
692                         $(SRC)/common/mapfiles/gen/sparcv9_gcc_map.noexeglobs
693 MAPFILE.NGB_i386=       $(SRC)/common/mapfiles/gen/i386_cc_map.noexeglobs
694 $(__GNUC64)MAPFILE.NGB_i386= \
695                         $(SRC)/common/mapfiles/gen/i386_gcc_map.noexeglobs
696 MAPFILE.NGB_amd64=      $(SRC)/common/mapfiles/gen/amd64_cc_map.noexeglobs
697 $(__GNUC64)MAPFILE.NGB_amd64= \
698                         $(SRC)/common/mapfiles/gen/amd64_gcc_map.noexeglobs
699 MAPFILE.NGB =           $(MAPFILE.NGB_$(MACH))

701 #
702 # A generic interface mapfile name, used by various dynamic objects to define
703 # the interfaces and interposers the object must export.
704 #
705 MAPFILE.INT =           mapfile-intf

707 #
708 # LDLIBS32 and LDLIBS64 can be set in the environment to override the following
709 # assignments.
710 #
711 # These environment settings make sure that no libraries are searched outside
712 # of the local workspace proto area:
713 #       LDLIBS32=-YP,$ROOT/lib:$ROOT/usr/lib
714 #       LDLIBS64=-YP,$ROOT/lib/$MACH64:$ROOT/usr/lib/$MACH64
715 #
716 LDLIBS32 =      $(ENVLDLIBS1) $(ENVLDLIBS2) $(ENVLDLIBS3)
717 LDLIBS32 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib -L%/lib)
718 LDLIBS.cmd =    $(LDLIBS32)
719 LDLIBS.lib =    $(LDLIBS32)
```



```
721 LDLIBS64 =      $(ENVLDLIBS1:%=%/$(MACH64)) \
722                 $(ENVLDLIBS2:%=%/$(MACH64)) \
723                 $(ENVLDLIBS3:%=%/$(MACH64))
724 LDLIBS64 +=     $(ADJUNCT_PROTO:%=-L%/usr/lib/$(MACH64) -L%/lib/$(MACH64))

726 #
727 # Define compilation macros.
728 #
729 COMPILE.c=      $(CC) $(CFLAGS) $(CPPFLAGS) -c
730 COMPILE64.c=    $(CC) $(CFLAGS64) $(CPPFLAGS) -c
731 COMPILE.cc=     $(CCC) $(CCFLAGS) $(CPPFLAGS) -c
732 COMPILE64.cc=   $(CCC) $(CCFLAGS64) $(CPPFLAGS) -c
733 COMPILE.s=      $(AS) $(ASFLAGS) $(AS_CPPFLAGS)
734 COMPILE64.s=    $(AS) $(ASFLAGS) $($(MACH64)_AS_XARCH) $(AS_CPPFLAGS)
735 COMPILE.d=      $(DTRACE) -G -32
736 COMPILE64.d=    $(DTRACE) -G -64
737 COMPILE.b=      $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))
738 COMPILE64.b=    $(ELFWRAP) $(ELFWRAP_FLAGS$(CLASS))

740 CLASSPATH=      .
741 COMPILE.java=   $(JAVAC) $(JAVAFLAGS) -classpath $(CLASSPATH)

743 #
744 # Link time macros
745 #
746 CCNEEDED                = -lC
747 CCEXTNEEDED             = -lCrun -lCstd
748 $(__GNUC)CCNEEDED       = -L$(GCCLIBDIR) -lstdc++ -lgcc_s
749 $(__GNUC)CCEXTNEEDED    = $(CCNEEDED)

751 LINK.c=         $(CC) $(CFLAGS) $(CPPFLAGS) $(LDFLAGS)
752 LINK64.c=       $(CC) $(CFLAGS64) $(CPPFLAGS) $(LDFLAGS)
753 NORUNPATH=      -norunpath -nolib
754 LINK.cc=        $(CCC) $(CCFLAGS) $(CPPFLAGS) $(NORUNPATH) \
755                 $(LDFLAGS) $(CCNEEDED)
756 LINK64.cc=      $(CCC) $(CCFLAGS64) $(CPPFLAGS) $(NORUNPATH) \
757                 $(LDFLAGS) $(CCNEEDED)

759 #
760 # lint macros
761 #
762 # Note that the undefine of __PRAGMA_REDEFINE_EXTNAME can be removed once
763 # ON is built with a version of lint that has the fix for 4484186.
764 #
765 ALWAYS_LINT_DEFS =      -errtags=yes -s
766 ALWAYS_LINT_DEFS +=     -erroff=E_PTRDIFF_OVERFLOW
767 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_NARROW_CONV
768 ALWAYS_LINT_DEFS +=     -U__PRAGMA_REDEFINE_EXTNAME
769 ALWAYS_LINT_DEFS +=     $(C99LMODE)
770 ALWAYS_LINT_DEFS +=     -errsecurity=$(SECLEVEL)
771 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_CREAT_WITHOUT_EXCL
772 ALWAYS_LINT_DEFS +=     -erroff=E_SEC_FORBIDDEN_WARN_CREAT
773 # XX64 -- really only needed for amd64 lint
774 ALWAYS_LINT_DEFS +=     -erroff=E_ASSIGN_INT_TO_SMALL_INT
775 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_CONST_TO_SMALL_INT
776 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_INT_TO_SMALL_INT
777 ALWAYS_LINT_DEFS +=     -erroff=E_CAST_TO_PTR_FROM_INT
778 ALWAYS_LINT_DEFS +=     -erroff=E_COMP_INT_WITH_LARGE_INT
779 ALWAYS_LINT_DEFS +=     -erroff=E_INTEGRAL_CONST_EXP_EXPECTED
780 ALWAYS_LINT_DEFS +=     -erroff=E_PASS_INT_TO_SMALL_INT
781 ALWAYS_LINT_DEFS +=     -erroff=E_PTR_CONV_LOSES_BITS

783 # This forces lint to pick up note.h and sys/note.h from Devpro rather than
784 # from the proto area.  The note.h that ON delivers would disable NOTE().
785 ONLY_LINT_DEFS =        -I$(SPRO_VROOT)/prod/include/lint
```

```
787 SECLEVEL=       core
788 LINT.c=         $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS) $(CPPFLAGS) \
789                 $(ALWAYS_LINT_DEFS)
790 LINT64.c=       $(LINT) $(ONLY_LINT_DEFS) $(LINTFLAGS64) $(CPPFLAGS) \
791                 $(ALWAYS_LINT_DEFS)
792 LINT.s=         $(LINT.c)

794 # For some future builds, NATIVE_MACH and MACH might be different.
795 # Therefore, NATIVE_MACH needs to be redefined in the
796 # environment as 'uname -p' to override this macro.
797 #
798 # For now at least, we cross-compile amd64 on i386 machines.
799 NATIVE_MACH=    $(MACH:amd64=i386)

801 # Define native compilation macros
802 #

804 # Base directory where compilers are loaded.
805 # Defined here so it can be overridden by developer.
806 #
807 SPRO_ROOT=              $(BUILD_TOOLS)/SUNWspro
808 SPRO_VROOT=             $(SPRO_ROOT)/SS12
809 GNU_ROOT=               /usr

811 $(__GNUC)PRIMARY_CC=    gcc4,$(GNUC_ROOT)/bin/gcc,gnu
812 $(__SUNC)PRIMARY_CC=    studio12,$(SPRO_VROOT)/bin/cc,sun
813 $(__GNUC)PRIMARY_CCC=   gcc4,$(GNUC_ROOT)/bin/g++,gnu
814 $(__SUNC)PRIMARY_CCC=   studio12,$(SPRO_VROOT)/bin/CC,sun

816 CW_CC_COMPILERS=        $(PRIMARY_CC:%=--primary %) $(SHADOW_CCS:%=--shadow %)
817 CW_CCC_COMPILERS=       $(PRIMARY_CCC:%=--primary %) $(SHADOW_CCCS:%=--shadow %)


820 # Till SS12u1 formally becomes the NV CBE, LINT is hard
821 # coded to be picked up from the $SPRO_ROOT/sunstudio12.1/
822 # location. Impacted variables are sparc_LINT, sparcv9_LINT,
823 # i386_LINT, amd64_LINT.
824 # Reset them when SS12u1 is rolled out.
825 #

827 # Specify platform compiler versions for languages
828 # that we use (currently only c and c++).
829 #
830 CW=                     $(ONBLD_TOOLS)/bin/$(MACH)/cw

832 BUILD_CC=               $(CW) $(CW_CC_COMPILERS) --
833 BUILD_CCC=              $(CW) -C $(CW_CCC_COMPILERS) --
834 BUILD_CPP=              /usr/ccs/lib/cpp
835 BUILD_LD=               /usr/ccs/bin/ld
836 BUILD_LINT=             $(SPRO_ROOT)/sunstudio12.1/bin/lint

838 $(MACH)_CC=             $(BUILD_CC)
839 $(MACH)_CCC=            $(BUILD_CCC)
840 $(MACH)_CPP=            $(BUILD_CPP)
841 $(MACH)_LD=             $(BUILD_LD)
842 $(MACH)_LINT=           $(BUILD_LINT)
843 $(MACH64)_CC=           $(BUILD_CC)
844 $(MACH64)_CCC=          $(BUILD_CCC)
845 $(MACH64)_CPP=          $(BUILD_CPP)
846 $(MACH64)_LD=           $(BUILD_LD)
847 $(MACH64)_LINT=         $(BUILD_LINT)

849 sparc_AS=               /usr/ccs/bin/as -xregsym=no
850 sparcv9_AS=             $($(MACH)_AS)

852 i386_AS=                /usr/ccs/bin/as
```



```
853 $(__GNUC)i386_AS=       $(ONBLD_TOOLS)/bin/$(MACH)/aw
854 amd64_AS=               $(ONBLD_TOOLS)/bin/$(MACH)/aw

856 NATIVECC=               $($(NATIVE_MACH)_CC)
857 NATIVECCC=              $($(NATIVE_MACH)_CCC)
858 NATIVECPP=              $($(NATIVE_MACH)_CPP)
859 NATIVEAS=               $($(NATIVE_MACH)_AS)
860 NATIVELD=               $($(NATIVE_MACH)_LD)
861 NATIVELINT=             $($(NATIVE_MACH)_LINT)

863 #
864 # Makefile.master.64 overrides these settings
865 #
866 CC=                     $(NATIVECC)
867 CCC=                    $(NATIVECCC)
868 CPP=                    $(NATIVECPP)
869 AS=                     $(NATIVEAS)
870 LD=                     $(NATIVELD)
871 LINT=                   $(NATIVELINT)

873 # Pass -Y flag to cpp (method of which is release-dependent)
874 CCYFLAG=                -Y I,

876 BDIRECT=        -Bdirect
877 BDYNAMIC=       -Bdynamic
878 BLOCAL=         -Blocal
879 BNODIRECT=      -Bnodirect
880 BREDUCE=        -Breduce
881 BSTATIC=        -Bstatic

883 ZDEFS=          -zdefs
884 ZDIRECT=        -zdirect
885 ZIGNORE=        -zignore
886 ZINITFIRST=     -zinitfirst
887 ZINTERPOSE=     -zinterpose
888 ZLAZYLOAD=      -zlazyload
889 ZLOADFLTR=      -zloadfltr
890 ZMULDEFS=       -zmuldefs
891 ZNODEFAULTLIB=  -znodefaultlib
892 ZNODEFS=        -znodefs
893 ZNODELETE=      -znodelete
894 ZNODLOPEN=      -znodlopen
895 ZNODUMP=        -znodump
896 ZNOLAZYLOAD=    -znolazyload
897 ZNOLDYNSYM=     -znoldynsym
898 ZNORELOC=       -znoreloc
899 ZNOVERSION=     -znoversion
900 ZRECORD=        -zrecord
901 ZREDLOCSYM=     -zredlocsym
902 ZTEXT=          -ztext
903 ZVERBOSE=       -zverbose

905 GSHARED=        -G
906 CCMT=           -mt

908 # Handle different PIC models on different ISAs
909 # (May be overridden by lower-level Makefiles)

911 sparc_C_PICFLAGS =      -fpic
912 sparcv9_C_PICFLAGS =    -fpic
913 i386_C_PICFLAGS =       -fpic
914 amd64_C_PICFLAGS =      -fpic
915 C_PICFLAGS =            $($(MACH)_C_PICFLAGS)
916 C_PICFLAGS64 =          $($(MACH64)_C_PICFLAGS)

918 sparc_C_BIGPICFLAGS =   -fPIC
```

```
 919 sparcv9_C_BIGPICFLAGS = -fPIC
 920 i386_C_BIGPICFLAGS =    -fPIC
 921 amd64_C_BIGPICFLAGS =   -fPIC
 922 C_BIGPICFLAGS =         $($(MACH)_C_BIGPICFLAGS)
 923 C_BIGPICFLAGS64 =       $($(MACH64)_C_BIGPICFLAGS)

 925 # CC requires there to be no space between '-K' and 'pic' or 'PIC'.
 926 # and does not support -f
 927 sparc_CC_PICFLAGS =     -_cc=-Kpic -_gcc=-fpic
 928 sparcv9_CC_PICFLAGS =   -_cc=-KPIC -_gcc=-fPIC
 929 i386_CC_PICFLAGS =      -_cc=-Kpic -_gcc=-fpic
 930 amd64_CC_PICFLAGS =     -_cc=-Kpic -_gcc=-fpic
 931 CC_PICFLAGS =           $($(MACH)_CC_PICFLAGS)
 932 CC_PICFLAGS64 =         $($(MACH64)_CC_PICFLAGS)

 934 AS_PICFLAGS=            -K pic
 935 AS_BIGPICFLAGS=         -K PIC

 937 #
 938 # Default label for CTF sections
 939 #
 940 CTFCVTFLAGS=            -i -L VERSION

 942 #
 943 # Override to pass module-specific flags to ctfmerge.  Currently used only by
 944 # krtld to turn on fuzzy matching, and source-level debugging to inhibit
 945 # stripping.
 946 #
 947 CTFMRGFLAGS=

 949 CTFCONVERT_O            = $(CTFCONVERT) $(CTFCVTFLAGS) $@

 951 # Rules (normally from make.rules) and macros which are used for post
 952 # processing files. Normally, these do stripping of the comment section
 953 # automatically.
 954 #    RELEASE_CM:        Should be editted to reflect the release.
 955 #    POST_PROCESS_O:    Post-processing for '.o' files.
 956 #    POST_PROCESS_A:    Post-processing for '.a' files (currently null).
 957 #    POST_PROCESS_SO:   Post-processing for '.so' files.
 958 #    POST_PROCESS:      Post-processing for executable files (no suffix).
 959 # Note that these macros are not completely generalized as they are to be
 960 # used with the file name to be processed following.
 961 #
 962 # It is left as an exercise to Release Engineering to embellish the generation
 963 # of the release comment string.
 964 #
 965 #       If this is a standard development build:
 966 #               compress the comment section (mcs -c)
 967 #               add the standard comment (mcs -a $(RELEASE_CM))
 968 #               add the development specific comment (mcs -a $(DEV_CM))
 969 #
 970 #       If this is an installation build:
 971 #               delete the comment section (mcs -d)
 972 #               add the standard comment (mcs -a $(RELEASE_CM))
 973 #               add the development specific comment (mcs -a $(DEV_CM))
 974 #
 975 #       If this is an release build:
 976 #               delete the comment section (mcs -d)
 977 #               add the standard comment (mcs -a $(RELEASE_CM))
 978 #
 979 # The following list of macros are used in the definition of RELEASE_CM
 980 # which is used to label all binaries in the build:
 981 #
 982 #       RELEASE         Specific release of the build, eg: 5.2
 983 #       RELEASE_MAJOR   Major version number part of $(RELEASE)
 984 #       RELEASE_MINOR   Minor version number part of $(RELEASE)
```



```
 985 #       VERSION         Version of the build (alpha, beta, Generic)
 986 #       PATCHID         If this is a patch this value should contain
 987 #                       the patchid value (eg: "Generic 100832-01"), otherwise
 988 #                       it will be set to $(VERSION)
 989 #       RELEASE_DATE    Date of the Release Build
 990 #       PATCH_DATE      Date the patch was created, if this is blank it
 991 #                       will default to the RELEASE_DATE
 992 #
 993 RELEASE_MAJOR=  5
 994 RELEASE_MINOR=  11
 995 RELEASE=        $(RELEASE_MAJOR).$(RELEASE_MINOR)
 996 VERSION=        SunOS Development
 997 PATCHID=        $(VERSION)
 998 RELEASE_DATE=   release date not set
 999 PATCH_DATE=     $(RELEASE_DATE)
1000 RELEASE_CM=     "@($(POUND_SIGN))SunOS $(RELEASE) $(PATCHID) $(PATCH_DATE)"
1001 DEV_CM=         "@($(POUND_SIGN))SunOS Internal Development: non-nightly build"

1003 PROCESS_COMMENT=  @?${MCS} -d -a $(RELEASE_CM) -a $(DEV_CM)
1004 $(RELEASE_BUILD)PROCESS_COMMENT=   @?${MCS} -d -a $(RELEASE_CM)

1006 STRIP_STABS=                       $(STRIP) -x $@
1007 $(SRCDBGBLD)STRIP_STABS=           :

1009 POST_PROCESS_O=
1010 POST_PROCESS_A=
1011 POST_PROCESS_SO=        $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
1012                         $(ELFSIGN_OBJECT)
1013 POST_PROCESS=           $(PROCESS_COMMENT) $@ ; $(STRIP_STABS) ; \
1014                         $(ELFSIGN_OBJECT)

1016 #
1017 # chk4ubin is a tool that inspects a module for a symbol table
1018 # ELF section size which can trigger an OBP bug on older platforms.
1019 # This problem affects only specific sun4u bootable modules.
1020 #
1021 CHK4UBIN=       $(ONBLD_TOOLS)/bin/$(MACH)/chk4ubin
1022 CHK4UBINFLAGS=
1023 CHK4UBINARY=    $(CHK4UBIN) $(CHK4UBINFLAGS) $@

1025 #
1026 # PKGARCHIVE specifies the default location where packages should be
1027 # placed if built.
1028 #
1029 $(RELEASE_BUILD)PKGARCHIVESUFFIX=       -nd
1030 PKGARCHIVE=$(SRC)/../../packages/$(MACH)/nightly$(PKGARCHIVESUFFIX)

1032 #
1033 # The repositories will be created with these publisher settings.  To
1034 # update an image to the resulting repositories, this must match the
1035 # publisher name provided to "pkg set-publisher."
1036 #
1037 PKGPUBLISHER_REDIST=    on-nightly
1038 PKGPUBLISHER_NONREDIST= on-extra

1040 #       Default build rules which perform comment section post-processing.
1041 #
1042 .c:
1043         $(LINK.c) -o $@ $< $(LDLIBS)
1044         $(POST_PROCESS)
1045 .c.o:
1046         $(COMPILE.c) $(OUTPUT_OPTION) $< $(CTFCONVERT_HOOK)
1047         $(POST_PROCESS_O)
1048 .c.a:
1049         $(COMPILE.c) -o $% $<
1050         $(PROCESS_COMMENT) $%
```

```
1051         $(AR) $(ARFLAGS) $@ $%
1052         $(RM) $%
1053 .s.o:
1054         $(COMPILE.s) -o $@ $<
1055         $(POST_PROCESS_O)
1056 .s.a:
1057         $(COMPILE.s) -o $% $<
1058         $(PROCESS_COMMENT) $%
1059         $(AR) $(ARFLAGS) $@ $%
1060         $(RM) $%
1061 .cc:
1062         $(LINK.cc) -o $@ $< $(LDLIBS)
1063         $(POST_PROCESS)
1064 .cc.o:
1065         $(COMPILE.cc) $(OUTPUT_OPTION) $<
1066         $(POST_PROCESS_O)
1067 .cc.a:
1068         $(COMPILE.cc) -o $% $<
1069         $(AR) $(ARFLAGS) $@ $%
1070         $(PROCESS_COMMENT) $%
1071         $(RM) $%
1072 .y:
1073         $(YACC.y) $<
1074         $(LINK.c) -o $@ y.tab.c $(LDLIBS)
1075         $(POST_PROCESS)
1076         $(RM) y.tab.c
1077 .y.o:
1078         $(YACC.y) $<
1079         $(COMPILE.c) -o $@ y.tab.c $(CTFCONVERT_HOOK)
1080         $(POST_PROCESS_O)
1081         $(RM) y.tab.c
1082 .l:
1083         $(RM) $*.c
1084         $(LEX.l) $< > $*.c
1085         $(LINK.c) -o $@ $*.c -ll $(LDLIBS)
1086         $(POST_PROCESS)
1087         $(RM) $*.c
1088 .l.o:
1089         $(RM) $*.c
1090         $(LEX.l) $< > $*.c
1091         $(COMPILE.c) -o $@ $*.c $(CTFCONVERT_HOOK)
1092         $(POST_PROCESS_O)
1093         $(RM) $*.c

1095 .bin.o:
1096         $(COMPILE.b) -o $@ $<
1097         $(POST_PROCESS_O)

1099 .java.class:
1100         $(COMPILE.java) $<

1102 # Bourne and Korn shell script message catalog build rules.
1103 # We extract all gettext strings with sed(1) (being careful to permit
1104 # multiple gettext strings on the same line), weed out the dups, and
1105 # build the catalogue with awk(1).

1107 .sh.po .ksh.po:
1108         $(SED) -n -e ":a"                                           \
1109                   -e "h"                                            \
1110                   -e "s/.*gettext *\(\"[^\"]*\"\).*/\1/p"           \
1111                   -e "x"                                            \
1112                   -e "s/\(.*\)gettext *\"[^\"]*\"\(.*\)/\1\2/"      \
1113                   -e "t a"                                          \
1114                 $< | sort -u | $(AWK) '{ print "msgid\t" $$0 "\nmsgstr" }' > $@

1116 #
```



```
1117 # Python and Perl executable and message catalog build rules.
1118 #
1119 .SUFFIXES: .pl .pm .py .pyc

1121 .pl:
1122         $(RM) $@;
1123         $(SED) -e "s@TEXT_DOMAIN@\"$(TEXT_DOMAIN)\"@" $< > $@;
1124         $(CHMOD) +x $@

1126 .py:
1127         $(RM) $@; $(SED) -e "1s:^\#!@PYTHON@:\#!$(PYTHON):" < $< > $@; $(CHMOD)

1129 .py.pyc:
1130         $(RM) $@
1131         $(PYTHON) -mpy_compile $<
1132         @[ $(<)c = $@ ] || $(MV) $(<)c $@

1134 .py.po:
1135         $(GNUXGETTEXT) $(GNUXGETFLAGS) -d $(<F:%.py=%) $< ;

1137 .pl.po .pm.po:
1138         $(XGETTEXT) $(XGETFLAGS) -d $(<F) $< ;
1139         $(RM)   $@ ;
1140         $(SED) "/^domain/d" < $(<F).po > $@ ;
1141         $(RM) $(<F).po

1143 #
1144 # When using xgettext, we want messages to go to the default domain,
1145 # rather than the specified one.  This special version of the
1146 # COMPILE.cpp macro effectively prevents expansion of TEXT_DOMAIN,
1147 # causing xgettext to put all messages into the default domain.
1148 #
1149 CPPFORPO=$(COMPILE.cpp:\"$(TEXT_DOMAIN)\"=TEXT_DOMAIN)

1151 .c.i:
1152         $(CPPFORPO) $< > $@

1154 .h.i:
1155         $(CPPFORPO) $< > $@

1157 .y.i:
1158         $(YACC) -d $<
1159         $(CPPFORPO) y.tab.c  > $@
1160         $(RM) y.tab.c

1162 .l.i:
1163         $(LEX) $<
1164         $(CPPFORPO) lex.yy.c  > $@
1165         $(RM) lex.yy.c

1167 .c.po:
1168         $(CPPFORPO) $< > $<.i
1169         $(BUILD.po)

1171 .cc.po:
1172         $(CPPFORPO) $< > $<.i
1173         $(BUILD.po)

1175 .y.po:
1176         $(YACC) -d $<
1177         $(CPPFORPO) y.tab.c  > $<.i
1178         $(BUILD.po)
1179         $(RM) y.tab.c

1181 .l.po:
1182         $(LEX) $<
```

```
1183         $(CPPFORPO) lex.yy.c  > $<.i
1184         $(BUILD.po)
1185         $(RM) lex.yy.c

1187 #
1188 # Rules to perform stylistic checks
1189 #
1190 .SUFFIXES: .x .xml .check .xmlchk

1192 .h.check:
1193         $(DOT_H_CHECK)

1195 .x.check:
1196         $(DOT_X_CHECK)

1198 .xml.xmlchk:
1199         $(MANIFEST_CHECK)

1201 #
1202 # Include rules to render automated sccs get rules "safe".
1203 #
1204 include $(SRC)/Makefile.noget
```

```
**********************************************************
    1123 Fri Dec 21 14:59:54 2018
new/usr/src/Makefile.smatch
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#
# Copyright 2018 Joyent, Inc.
#

#
# smatch/sparse checks we always disable, due to too many false positives (or
# simply too much legacy).
#

SMATCH_ARGS = --disable=uninitialized,check_check_deref,unreachable

# VLAs are OK by us
SMATCH_ARGS += -Wno-vla
# don't care
SMATCH_ARGS += -Wno-one-bit-signed-bitfield
# there are lots of "extern void myfunc() { ... }" around
SMATCH_ARGS += -Wno-external-function-has-definition
# we have lots of legacy "void foo();" in headers
SMATCH_ARGS += -Wno-old-style-definition
SMATCH_ARGS += -Wno-strict-prototypes

CERRWARN += $(SMATCH_ARGS:%=-_smatch=%)

CERRWARN += $(SMOFF:%=-_smatch=--disable=%)

SMATCH_ = 
SMATCH_on =
SMATCH_off = -_smatch=off

CERRWARN += $(SMATCH_$(SMATCH))
```

```
**********************************************************
    5017 Fri Dec 21 14:59:54 2018
new/usr/src/cmd/bnu/Makefile.inc
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.
#

SRCS =          account.c anlwrk.c bnuconvert.c callers.c               \
                chremdir.c cntrl.c conn.c cpmv.c ct.c                   \
                cu.c dial.c dio.c dkbreak.c                             \
                dkdial.c dkerr.c dkminor.c dtnamer.c eio.c              \
                expfile.c fio.c gename.c getargs.c getopt.c getprm.c    \
                getpwinfo.c gio.c gnamef.c gnxseq.c grades.c            \
                gtcfile.c gwd.c imsg.c in.uucpd.c interface.c limits.c  \
                line.c logent.c mailst.c perfstat.c                     \
                permission.c pk0.c pk1.c pkdefs.c security.c            \
                setmode.c shio.c statlog.c stoa.c strpbrk.c     \
                strsave.c sysfiles.c systat.c ulockf.c unknown.c        \
                utility.c uucheck.c uucico.c uucleanup.c uucp.c         \
                uucpdefs.c uucpname.c uudecode.c uuencode.c             \
                uuglist.c uuname.c uusched.c uustat.c uux.c             \
                uuxqt.c versys.c xio.c xqt.c

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-char-subscripts
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-extra
CERRWARN        += -_gcc=-Wno-implicit-function-declaration

# "parse error: parsing (arg (9223372034707292160-0,2-s32max) == 2)"
SMATCH          = off

TLILIB =        -lnsl -lsocket
PAMLIB =        -lpam

# use this if you don't have strpbrk/getopt in libc
# STRPBRK =     strpbrk.o
# GETOPT =      getopt.o
```



```
PROTOCOLOBJS =  dio.o eio.o gio.o xio.o fio.o

OTHEROBJS =     utility.o cpmv.o expfile.o gename.o getpwinfo.o \
                ulockf.o xqt.o logent.o gnamef.o systat.o       \
                sysfiles.o strsave.o $(GETOPT)

uucp \
uucp.cat :=     POBJS = uucpdefs.o uucp.o gwd.o permission.o getargs.o  \
                        getprm.o uucpname.o versys.o gtcfile.o grades.o \
                        $(STRPBRK) chremdir.o mailst.o $(OTHEROBJS)

uux \
uux.cat :=      POBJS = uucpdefs.o uux.o mailst.o gwd.o permission.o    \
                        getargs.o getprm.o uucpname.o versys.o gtcfile.o \
                        grades.o chremdir.o $(STRPBRK) $(OTHEROBJS)

uuxqt \
uuxqt.cat :=    POBJS = uucpdefs.o uuxqt.o mailst.o getprm.o uucpname.o \
                        permission.o getargs.o gtcfile.o grades.o       \
                        $(STRPBRK) shio.o chremdir.o account.o          \
                        perfstat.o statlog.o security.o limits.o        \
                        $(OTHEROBJS)

uucico \
uucico.cat :=   POBJS = uucpdefs.o uucico.o conn.o callers.o cntrl.o    \
                        pk0.o pk1.o anlwrk.o permission.o getargs.o     \
                        gnxseq.o pkdefs.o imsg.o gtcfile.o grades.o     \
                        mailst.o uucpname.o line.o chremdir.o           \
                        interface.o statlog.o stoa.o                    \
                        perfstat.o account.o security.o limits.o        \
                        $(STRPBRK) $(PROTOCOLOBJS) $(OTHEROBJS)         \
                        versys.o setmode.o
uucico:=        PLIBS = $(TLILIB) -lgen

uuname \
uuname.cat :=   POBJS = uuname.o uucpname.o uucpdefs.o getpwinfo.o      \
                        sysfiles.o strsave.o getargs.o

uustat \
uustat.cat :=   POBJS = uustat.o gnamef.o expfile.o uucpdefs.o          \
                        getpwinfo.o ulockf.o getargs.o utility.o        \
                        uucpname.o versys.o strsave.o sysfiles.o        \
                        cpmv.o mailst.o permission.o $(STRPBRK) \
                        $(GETOPT)

uusched \
uusched.cat :=  POBJS = uusched.o gnamef.o expfile.o uucpdefs.o         \
                        getpwinfo.o ulockf.o systat.o getargs.o         \
                        utility.o limits.o permission.o uucpname.o      \
                        $(GETOPT)

uucleanup \
uucleanup.cat := POBJS = uucleanup.o gnamef.o expfile.o uucpdefs.o      \
                        getpwinfo.o uucpname.o ulockf.o getargs.o       \
                        cpmv.o utility.o $(GETOPT)

uuglist \
uuglist.cat :=  POBJS = grades.o cpmv.o getargs.o getpwinfo.o strsave.o \
                        uuglist.o uucpdefs.o expfile.o uucpname.o $(GETOPT)

bnuconvert \
bnuconvert.cat := POBJS = bnuconvert.o uucpdefs.o grades.o strsave.o    \
                        getpwinfo.o getargs.o cpmv.o chremdir.o         \
                        expfile.o gename.o gnamef.o gtcfile.o logent.o  \
                        systat.o ulockf.o utility.o uucpname.o $(GETOPT)
```

```
127 remote.unknown \
128 remote.unknown.cat := POBJS =   unknown.o

130 cu \
131 cu.cat :=       POBJS = cu.o callers.o getargs.o line.o uucpdefs.o      \
132                         ulockf.o conn.o interface.o strsave.o           \
133                         sysfiles.o stoa.o setmode.o
134 cu:=            PLIBS = $(TLILIB) -lgen

136 ct \
137 ct.cat :=       POBJS = ct.o callers.o getargs.o line.o uucpdefs.o      \
138                         ulockf.o conn.o interface.o sysfiles.o          \
139                         strsave.o stoa.o setmode.o
140 ct:=            PLIBS = $(TLILIB) -lgen

142 uudecode \
143 uudecode.cat := POBJS = uudecode.o common.o

145 uuencode \
146 uuencode.cat := POBJS = uuencode.o

148 uucheck \
149 uucheck.cat :=  POBJS = uucheck.o uucpname.o getargs.o $(GETOPT)

151 in.uucpd \
152 in.uucpd.cat := POBJS = in.uucpd.o
153 in.uucpd:=      PLIBS = $(TLILIB) $(PAMLIB)
```

```
**********************************************************
    3107 Fri Dec 21 14:59:54 2018
new/usr/src/cmd/cmd-inet/usr.bin/pppd/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# cmd/cmd-inet/usr.bin/pppd/Makefile
#
# Copyright (c) 2018, Joyent, Inc.

include         ../../../Makefile.cmd
include         Makefile.def

PROG=           pppd
SUBDIRS=        plugins
OBJS=           auth.o ccp.o chap.o demand.o fsm.o ipcp.o ipv6cp.o \
                lcp.o magic.o main.o options.o sys-solaris.o upap.o utils.o \
                multilink.o cbcp.o

# Object tdb.o used only for Multilink; not supported yet.

all:=           TARGET= all
install:=       TARGET= install
clean:=         TARGET= clean
clobber:=       TARGET= clobber
lint:=          TARGET= lint

LDLIBS +=       -lpam -lmd -lsocket -lnsl -ldlpi

#
# We need absolute path to /etc/ppp/plugins and /usr/lib/inet/ppp, not
# that of the proto area
#
LDFLAGS +=      -R$(ETCPPPPLUGINDIR_ABS) -R$(LIBPPPPLUGINDIR_ABS)

CPPFLAGS +=     -DPLUGIN -DSVR4 -DSOL2 -DINET6
CPPFLAGS +=     -D_PATH_VARRUN='"/var/run/"'
CPPFLAGS +=     -DNEGOTIATE_FCS -DCBCP_SUPPORT -DALLOW_PAM -DHAS_SHADOW
CPPFLAGS +=     -DHAVE_MMAP -DCOMP_TUNE -DMUX_FRAME
$(NOT_RELEASE_BUILD)CPPFLAGS += -DDEBUG

.KEEP_STATE:

.PARALLEL:      $(SUBDIRS)

all:            $(PROG) $(SUBDIRS)

# MS-CHAP support
CPPFLAGS +=     -DHAVE_CRYPT_H -DUSE_CRYPT -DHAVE_LIBMD
CPPFLAGS +=     -DCHAPMS -DMSLANMAN
CPPFLAGS +=     -DCHAPMSV2
OBJS +=         chap_ms.o
EXOBJS +=       mschap_test.o
CLOBBERFILES += mschap_test

CERRWARN +=     -_gcc=-Wno-uninitialized

# main() is too hairy for smatch
SMATCH=off

# This is used *only* for testing the portability of the libraries
# required for MS-CHAPv1.  It is not needed in any normal system and
```



```
# is not built by default.
mschap_test:    mschap_test.o chap_ms.o
        $(LINK.c) -o mschap_test mschap_test.o chap_ms.o $(LDFLAGS) -lmd
        @echo "Run with 'mschap_test 00000000000000000000000000000000 hello'"
        @echo
        @echo "Output should be:"
        @echo
        @echo " MS-CHAPv1 with LAN Manager -- 49 bytes:"
        @echo "  C9 CA EE 9B 1C A7 87 04"
        @echo "  79 36 8C 55 AB 88 EC 5A"
        @echo "  57 E9 A1 B7 95 40 C3 74"
        @echo "  F4 D9 9D AF 82 64 DC 3C"
        @echo "  53 F9 BC 92 14 B5 5D 9E"
        @echo "  78 C4 21 48 9D B7 A8 B4"
        @echo "  01"
        @echo "  MS-CHAPv2 -- 49 bytes:"
        @echo "  xx xx xx xx xx xx xx xx"
        @echo "  xx xx xx xx xx xx xx xx"
        @echo "  00 00 00 00 00 00 00 00"
        @echo "  xx xx xx xx xx xx xx xx"
        @echo "  xx xx xx xx xx xx xx xx"
        @echo "  xx xx xx xx xx xx xx xx"
        @echo "  00"

LINTFLAGS +=    -erroff=E_NAME_DEF_NOT_USED2

SRCS=           $(OBJS:%.o=%.c)

.PARALLEL:      $(OBJS)

$(PROG):        $(OBJS)
        $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

$(ROOTPROG):=   FILEMODE = 04555

ASPPP2PPPD=     $(ROOTUSRSBIN)/asppp2pppd
$(ASPPP2PPPD):= FILEMODE = 0550

install:        $(PROG) .WAIT $(SUBDIRS) $(ROOTPROG) $(ETCPPPDIR) $(ASPPP2PPPD)

$(ETCPPPDIR)/%: %
        $(INS.file)

$(ETCPPPDIR):
        $(INS.dir)

$(SUBDIRS):     FRC
        @cd $@; pwd; $(MAKE) $(TARGET)

FRC:

LINTOBJS=$(OBJS:%.o=%.ln)
CLOBBERFILES += $(LINTOBJS)

clean:          $(SUBDIRS)
        $(RM) $(OBJS) $(EXOBJS)

sha1.ln:=       LINTFLAGS += -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED

# Not using the default lint target here so that we can disable
# warnings per module as needed.
lint:           $(SUBDIRS) $(LINTOBJS)
        $(LINT.c) $(LINTOBJS) $(LDLIBS)

clobber:        $(SUBDIRS)
```

```
 128 include         ../../../Makefile.targ
```

```
**********************************************************
    1196 Fri Dec 21 14:59:55 2018
new/usr/src/cmd/deroff/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License, Version 1.0 only
# (the "License").  You may not use this file except in compliance
# with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 1989 by Sun Microsystems, Inc.
#
# Copyright (c) 2018, Joyent, Inc.

PROG= deroff

include ../Makefile.cmd

CERRWARN += -_gcc=-Wno-char-subscripts
CERRWARN += -_gcc=-Wno-unused-label
CERRWARN += -_gcc=-Wno-parentheses

# too hairy for smatch
SMATCH=off

.KEEP_STATE:

all: $(PROG)

install: all $(ROOTPROG)

clean:

lint:   lint_PROG

include ../Makefile.targ
```

***********************************************************
***1197 Fri Dec 21 14:59:55 2018***
***new/usr/src/cmd/devctl/Makefile***
***10063 basic support for smatch***
***10153 checkpaths shouldn't check packaging exceptions***
***********************************************************

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License, Version 1.0 only
# (the "License").  You may not use this file except in compliance
# with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# # Copyright (c) 2018, Joyent, Inc.

PROG= devctl

include ../Makefile.cmd

.KEEP_STATE:

CFLAGS +=       $(CCVERBOSE)
CERRWARN +=     -_gcc=-Wno-parentheses
LDLIBS += -ldevice -ldevinfo
LDLIBS += -ldevice -l devinfo

all: $(PROG)

install: all $(ROOTUSRSBINPROG)

clean:

lint:   lint_PROG

include ../Makefile.targ
```

```
**********************************************************
    1962 Fri Dec 21 14:59:55 2018
new/usr/src/cmd/fs.d/smbclnt/test/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
# Copyright 2013 Nexenta Systems, Inc.  All rights reserved.
# Copyright (c) 2018, Joyent, Inc.
#

include $(SRC)/cmd/Makefile.cmd

PROG=   srvenum srvinfo tconn
OBJS = $(PROG:%=%.o)
SRCS = $(OBJS:%.o=%.c)

# ROOTFS_PROG=  $(LIBPROG)
# include               ../../Makefile.fstype

ROOTOPTPKG = $(ROOT)/opt/smbcl-tests
TESTDIR = $(ROOTOPTPKG)/tests
INST_CMDS = $(PROG:%=$(TESTDIR)/%)

# OBJS= $(LIBPROG).o
# SRCS= $(LIBPROG).c $(FSLIBSRC)

CPPFLAGS += -I../../../../uts/common
CPPFLAGS += -I../../../../lib/libsmbfs

LDLIBS += -R'$$ORIGIN/../../../usr/lib'
LDLIBS += -R '$$ORIGIN/../../../usr/lib'
LDLIBS += -lsmbfs
LINTLIBS= -L$(ROOTLIB) -lsmbfs

CFLAGS += $(CCVERBOSE)
CERRWARN += -_gcc=-Wno-unused-variable
CSTD= $(CSTD_GNU99)

# not linted
SMATCH=off

LINTFLAGS += -erroff=E_FUNC_RET_ALWAYS_IGNOR2
```



```
# CLOBBERFILES  += $(LIBPROG)

all:    $(PROG)

install: all $(ROOTOPTPKG) $(TESTDIR) $(INST_CMDS)

lint:
        for f in $(SRCS); do ;\
        $(LINT.c) $$f $(LINTLIBS) ; done

clobber: clean
        -$(RM) $(PROG)

clean:
        -$(RM) $(OBJS)

$(ROOTOPTPKG):
        $(INS.dir)

$(TESTDIR):
        $(INS.dir)

$(TESTDIR)/%: %
        $(INS.file)

.KEEP_STATE:
```

```
**********************************************************
    1818 Fri Dec 21 14:59:55 2018
new/usr/src/cmd/fs.d/ufs/mkfs/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2006 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.
#

FSTYPE=         ufs
LIBPROG=        mkfs
ATTMK=          $(LIBPROG)

include         ../../Makefile.fstype
include         ../Makefile.roll

CPPFLAGS +=     -I../../

OBJS=           $(LIBPROG).o $(ROLLOBJS) $(FSLIB)
SRCS=           $(LIBPROG).c $(ROLLSRCS) $(FSLIBSRC)
MKFSOBJS=       mkfs.o

CERRWARN +=     -_gcc=-Wno-implicit-function-declaration
CERRWARN +=     -_gcc=-Wno-unused-variable
CERRWARN +=     -_gcc=-Wno-empty-body
CERRWARN +=     -_gcc=-Wno-uninitialized

# can't hack main() !
SMATCH =        off

# for messaging catalog
#
POFILE= mkfs.po

catalog:        $(POFILE)

$(POFILE):      $(SRCS)
        $(RM) $@
        $(COMPILE.cpp) $(SRCS)   > $(POFILE).i
        $(XGETTEXT) $(XGETFLAGS)        $(POFILE).i
        sed "/^domain/d" messages.po >  $@
        $(RM) $(POFILE).i messages.po
```



```
CPPFLAGS +=     -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64
LDLIBS +=       -ladm -lefi

$(LIBPROG):     $(OBJS)
                $(LINK.c) -o $@ $(OBJS) $(LDLIBS)
                $(POST_PROCESS)

lint:           lint_SRCS

clean:
        $(RM) $(MKFSOBJS)
```

```
**********************************************************
    4131 Fri Dec 21 14:59:55 2018
new/usr/src/cmd/geniconvtbl/Makefile.com
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.
#

$(NOT_NATIVE)NATIVE_BUILD = $(POUND_SIGN)

ITM     = geniconvtbl.so
PROG    = geniconvtbl

SRCSH1  = iconv_tm.h hash.h
SRCCH1  = itmcomp.h itm_util.h maptype.h
SRCSC1  = itmcomp.c assemble.c disassemble.c itm_util.c
SRCY1   = itm_comp.y
SRCL1   = itm_comp.l
SRCI1   = geniconvtbl.c


YTABC   = y.tab.c
YTABH   = y.tab.h
LEXYY   = lex.yy.c
YOUT    = y.output
MAPFILE = ../mapfile



SRCSH   = $(SRCSH1:%.h=../%.h)
SRCCH   = $(SRCCH1:%.h=../%.h)
SRCSC   = $(SRCSC1:%.c=../%.c)
SRCI    = $(SRCI1:%.c=../%.c)
SRCY    = $(SRCY1:%.y=../%.y)
SRCL    = $(SRCL1:%.l=../%.l)

SRCYC   = $(SRCY:%.y=%.c)
SRCLC   = $(SRCL:%.l=%.c)

SRCS    = $(SRCSC) $(YTABC) $(LEXYY)
HDRS    = $(SRCCH1) $(ERNOSTRH)
```



```
SED     = sed
LEXSED  = ../lex.sed
YACCSED = ../yacc.sed



# include ../../../lib/Makefile.lib
include ../../Makefile.cmd


ROOTDIRS32=     $(ROOTLIB)/iconv
ROOTDIRS64=     $(ROOTLIB)/iconv/$(MACH64)
ROOTITM32 =     $(ROOTDIRS32)/$(ITM)
ROOTITM64 =     $(ROOTDIRS64)/$(ITM)

#
# definition for some useful target like clean,
OBJS    = $(SRCSC1:%.c=%.o) $(YTABC:.c=.o) $(LEXYY:.c=.o)

CHECKHDRS = $(HDRS%.h=%.check)

CLOBBERFILES=   $(ITM) $(SRCYC)
CLEANFILES =    $(OBJS) $(YTABC) $(YTABH) $(LEXYY) $(YOUT) \
                $(POFILES) $(POFILE)

CPPFLAGS        += -I. -I..
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-unused-label
CERRWARN        += -_gcc=-Wno-switch
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-implicit-function-declaration
YFLAGS          += -d -v
CFLAGS          += -D_FILE_OFFSET_BITS=64

# dump_expr() is too hairy
SMATCH=off

$(ITM) :=       CFLAGS += $(GSHARED) $(C_PICFLAGS) $(ZTEXT) -h$@
$(ITM) :=       CFLAGS += $(GSHARED) $(C_PICFLAGS) $(ZTEXT) -h $@
$(ITM) :=       CPPFLAGS += -D_REENTRANT
$(ITM) :=       sparc_CFLAGS += -xregs=no%appl
$(ITM) :=       sparcv9_CFLAGS += -xregs=no%appl

LDLIBS += -lgen

MY_NATIVE_CPPFLAGS = -D_FILE_OFFSET_BITS=64 -I. -I..
MY_NATIVE_LDFLAGS = $(MAPFILE.NES:%=-M%) $(MAPFILE.PGA:%=-M%) $(MAPFILE.NED:%=-M
MY_NATIVE_LDLIBS = -lgen

#
# Message catalog
#
POFILES= $(SRCSC1:%.c=%.po) $(SRCI1:%.c=%.po) \
                $(SRCY1:%.y=%.po) $(SRCL1:%.l=%.po)

POFILE= geniconvtbl_.po




.KEEP_STATE:

.PARALLEL: $(ITM) $(OBJS)
```

```
$(PROG): $(OBJS)
        $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
        $(POST_PROCESS)

$(ITM): $(SRCI)
        $(CC) $(CFLAGS) $(CPPFLAGS) -M$(MAPFILE) -o $@ $(SRCI) $(LDLIBS)
        $(POST_PROCESS_SO)

$(YTABC) $(YTABH): $(SRCY)
        $(YACC) $(YFLAGS) $(SRCY)
        @ $(MV) $(YTABC) $(YTABC)~
        @ $(SED) -f $(YACCSED) $(YTABC)~ > $(YTABC)
        @ $(RM) $(YTABC)~

$(LEXYY): $(SRCL) $(YTABH)
        $(LEX) -t $(SRCL) | $(SED) -f $(LEXSED) > $(LEXYY)


$(POFILE):  .WAIT $(POFILES)
        $(RM) $@
        $(CAT) $(POFILES) >$@

$(POFILES): $(SRCSC) $(SRCI) $(SRCY) $(SRCL)

%.po:   ../%.c
        $(COMPILE.cpp) $<  > $<.i
        $(BUILD.po)


lint : lint_SRCS1  lint_SRCS2


lint_SRCS1: $(SRCS)
        $(LINT.c) $(SRCS) $(LDLIBS)

lint_SRCS2: $(SRCI)
        $(LINT.c) $(SRCI) $(LDLIBS)



hdrchk: $(HDRCHECKS)

cstyle: $(SRCS)
        $(DOT_C_CHECK)

clean:
        $(RM) $(CLEANFILES)

debug:
        $(MAKE) all COPTFLAG='' COPTFLAG64='' CFLAGS='-g -DDEBUG'


%.o:    %.c
        $(COMPILE.c) $<

%.o:    ../%.c
        $(COMPILE.c) $<



# install rule
#
$(ROOTDIRS32)/%: $(ROOTDIRS32) %
        -$(INS.file)
```



```
$(ROOTDIRS64)/%: $(ROOTDIRS64) %
        -$(INS.file)

$(ROOTDIRS32): $(ROOTLIB)
        -$(INS.dir)

$(ROOTDIRS64): $(ROOTDIRS32)
        -$(INS.dir)

$(ROOTLIB) $(ROOTBIN):
        -$(INS.dir)

include ../../Makefile.targ
```

```
**********************************************************
    1339 Fri Dec 21 14:59:55 2018
new/usr/src/cmd/ipcs/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License, Version 1.0 only
# (the "License").  You may not use this file except in compliance
# with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
#ident  "%Z%%M% %I%     %E% SMI"
#
# Copyright 2004 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.
# cmd/ipcs/Makefile
#

PROG=           ipcs
XPG4PROG=       $(PROG)

include ../Makefile.cmd

LINTFLAGS +=    -erroff=E_NAME_USED_NOT_DEF2
CFLAGS +=       $(CCVERBOSE)
LDLIBS +=       -lproject

# main() too hairy
SMATCH =        off

.KEEP_STATE:

all: $(PROG)

install: all $(ROOTXPG4PROG)

$(ROOTXPG4PROG):        $(ROOTPROG)
        -$(RM) $(ROOTXPG4PROG)
        -$(LN) -s ../../bin/$(PROG) $(ROOTXPG4PROG)

clean:

lint:   lint_PROG

include ../Makefile.targ
```

```
**********************************************************
    1584 Fri Dec 21 14:59:56 2018
new/usr/src/cmd/sendmail/Makefile.cmd
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.
#

CPPFLAGS.sm=    $(CPPFLAGS.master) -DSOLARIS=2$(RELEASE_MINOR)00 \
                -D_FILE_OFFSET_BITS=64
CERRWARN +=     -_gcc=-Wno-clobbered
CERRWARN +=     -_gcc=-Wno-parentheses
CERRWARN +=     -_gcc=-Wno-uninitialized
CERRWARN +=     -_gcc=-Wno-implicit-function-declaration
CERRWARN +=     -_gcc=-Wno-empty-body
CERRWARN +=     -_gcc=-Wno-unused-variable
CERRWARN +=     -_gcc=-Wno-unused-but-set-parameter
CERRWARN +=     -_gcc=-Wno-unused-but-set-variable
DBMDEF=         -DNDBM -DNEWDB -DNIS -DUSERDB -DMAP_REGEX -DLDAPMAP

# smatch can't handle main()
SMATCH =        off

ROOTLIBSMTPSM = $(ROOTLIB)/smtp/sendmail

$(ROOTLIBSMTPSM):
        $(INS.dir)

$(ROOTLIBSMTPSM)/%: $(ROOTLIBSMTPSM) %
        $(INS.file)
```

```
**********************************************************
    2354 Fri Dec 21 14:59:56 2018
new/usr/src/cmd/troff/nroff.d/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License, Version 1.0 only
# (the "License").  You may not use this file except in compliance
# with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright (c) 1989 by Sun Microsystems, Inc.
#
# Copyright (c) 2018, Joyent, Inc.

include         ../../Makefile.cmd

SUBDIRS =       terms.d

PROG =          nroff

OBJS =          n10.o n6.o

COMMONOBJS =    hytab.o n1.o n2.o n3.o n4.o n5.o        \
                n7.o n8.o n9.o ni.o nii.o suftab.o

SRCS =          $(OBJS:%.o=%.c) $(COMMONOBJS:%.o=../%.c)


CPPFLAGS =      -DNROFF -DUSG -DINCORE -DEUC -I. -I.. $(CPPFLAGS.master)

CERRWARN +=     -_gcc=-Wno-unused-value
CERRWARN +=     -_gcc=-Wno-extra
CERRWARN +=     -_gcc=-Wno-implicit-function-declaration
CERRWARN +=     -_gcc=-Wno-parentheses
CERRWARN +=     -_gcc=-Wno-unused-variable

# parse error: parsing (i (9223372034707292160-96) >= 65)
SMATCH =        off

LDLIBS +=       -lmapmalloc
#
# message catalog
#
POFILES= $(OBJS:%.o=%.po)
POFILE= nroff.d.po


all :=          TARGET= all
```



```
install :=      TARGET= install
clean :=        TARGET= clean
clobber :=      TARGET= clobber
lint :=         TARGET= lint
strip :=        TARGET= strip

# build rule for common source above
%.o: ../%.c
                $(COMPILE.c) $<

.KEEP_STATE:

.PARALLEL: $(COMMONOBJS) $(OBJS)

all :           $(PROG) $(SUBDIRS)

$(PROG) :       $(OBJS) $(COMMONOBJS)
                $(LINK.c) -o $@ $(OBJS) $(COMMONOBJS) $(LDLIBS)
                $(POST_PROCESS)

install :       $(PROG) $(ROOTPROG) $(SUBDIRS)

catalog:        $(POFILE)

$(POFILE):      $(POFILES)
         $(RM)  $@
         cat $(POFILES)  >       $@


clean :         $(SUBDIRS)
                $(RM) $(OBJS) $(COMMONOBJS)

strip :
                $(STRIP) $(PROG)

lint :          lint_SRCS

include         ../../Makefile.targ

# additional dependency for clobber which is defined in Makefile.targ
clobber :       $(SUBDIRS)

$(SUBDIRS) :    FRC
                @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
```

```
**********************************************************
    2775 Fri Dec 21 14:59:56 2018
new/usr/src/cmd/troff/troff.d/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# cmd/troff/troff.d/Makefile
#
# Copyright (c) 2018, Joyent, Inc.

include         ../../Makefile.cmd

PROG =          troff ta makedev

SUBDIRS =       tmac.d

TAOBJS =        draw.o ta.o
MAKEDEVOBJS =   makedev.o
TROFFOBJS =     t10.o t6.o
COMMONOBJS =    hytab.o n1.o n2.o n3.o n4.o n5.o                 \
                n7.o n8.o n9.o ni.o nii.o suftab.o

SRCS =          $(TAOBJS:%.o=%.c) $(MAKEDEVOBJS:%.o=%.c)        \
                $(TROFFOBJS:%.o=%.c) $(COMMONOBJS:%.o=../%.c)

TXTS =          README maketables troff.sh

CPPFLAGS =      -DUSG -DINCORE -I. -I.. $(CPPFLAGS.master)

CERRWARN +=     -_gcc=-Wno-implicit-function-declaration
CERRWARN +=     -_gcc=-Wno-unused-variable
CERRWARN +=     -_gcc=-Wno-parentheses
CERRWARN +=     -_gcc=-Wno-uninitialized
CERRWARN +=     -_gcc=-Wno-extra

# "parse error: parsing (i (9223372034707292160-96) >= 65)"
SMATCH =        off

#
# For message catalog
#
POFILES= $(TROFFOBJS:%.o=%.po) $(COMMONOBJS:%.o=../%.po)
```



```
POFILE= troff.d.po

# conditional assignments

all :=          TARGET= all
install :=      TARGET= install
clean :=        TARGET= clean
clobber :=      TARGET= clobber
lint :=         TARGET= lint
strip :=        TARGET= strip

troff:=         POBJS=  $(COMMONOBJS) $(TROFFOBJS)
ta:=            POBJS=  $(TAOBJS)
makedev:=       POBJS=  $(MAKEDEVOBJS)

ta:=            LDLIBS += -lm
troff:=         LDLIBS += -lmapmalloc

# build rule for common source above
%.o: ../%.c
                $(COMPILE.c) $<

.KEEP_STATE:

.PARALLEL: $(COMMONOBJS) $(TROFFOBJS) $(TAOBJS) $(MAKEDEVOBJS)

all :           $(PROG) $(TXTS) $(SUBDIRS)

$(PROG) :       $$(POBJS)
                $(LINK.c) -o $@ $(POBJS) $(LDLIBS)
                $(POST_PROCESS)

install:        $(PROG) $(ROOTPROG) $(SUBDIRS)

clean:          $(SUBDIRS)
                $(RM) $(TAOBJS) $(MAKEDEVOBJS) $(TROFFOBJS) $(COMMONOBJS)

catalog:        $(POFILE)

$(POFILE):      $(POFILES)
        $(RM)   $@
        cat     $(POFILES)      > $@


strip :
                $(STRIP $(PROG)

lint :          lint_SRCS

include         ../../Makefile.targ

# additional dependency for clobber which is defined in Makefile.targ
clobber:        $(SUBDIRS)

$(SUBDIRS) :    FRC
                @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
```

```
**********************************************************
   21123 Fri Dec 21 14:59:56 2018
new/usr/src/lib/libast/Makefile.com
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
#
# Copyright (c) 2018, Joyent, Inc.

SHELL=/usr/bin/ksh93

LIBRARY=        libast.a
VERS=           .1

# platform-independent sources are in common/
OBJECTS += \
        common/cdt/dtclose.o \
        common/cdt/dtdisc.o \
        common/cdt/dtextract.o \
        common/cdt/dtflatten.o \
        common/cdt/dthash.o \
        common/cdt/dtlist.o \
        common/cdt/dtmethod.o \
        common/cdt/dtnew.o \
        common/cdt/dtopen.o \
        common/cdt/dtrenew.o \
        common/cdt/dtrestore.o \
        common/cdt/dtsize.o \
        common/cdt/dtstat.o \
        common/cdt/dtstrhash.o \
        common/cdt/dttree.o \
        common/cdt/dttreeset.o \
        common/cdt/dtview.o \
        common/cdt/dtwalk.o \
        common/comp/atexit.o \
        common/comp/basename.o \
        common/comp/catopen.o \
        common/comp/closelog.o \
        common/comp/creat64.o \
        common/comp/dirname.o \
        common/comp/dup2.o \
        common/comp/eaccess.o \
        common/comp/errno.o \
```



```
        common/comp/execlp.o \
        common/comp/execve.o \
        common/comp/execvp.o \
        common/comp/execvpe.o \
        common/comp/fcntl.o \
        common/comp/fmtmsglib.o \
        common/comp/fnmatch.o \
        common/comp/frexp.o \
        common/comp/frexpl.o \
        common/comp/fsync.o \
        common/comp/ftw.o \
        common/comp/getdate.o \
        common/comp/getgroups.o \
        common/comp/getlogin.o \
        common/comp/getopt.o \
        common/comp/getoptl.o \
        common/comp/getpgrp.o \
        common/comp/getsubopt.o \
        common/comp/getwd.o \
        common/comp/gross.o \
        common/comp/hsearch.o \
        common/comp/iconv.o \
        common/comp/killpg.o \
        common/comp/link.o \
        common/comp/localeconv.o \
        common/comp/lstat.o \
        common/comp/memccpy.o \
        common/comp/memchr.o \
        common/comp/memcmp.o \
        common/comp/memcpy.o \
        common/comp/memmove.o \
        common/comp/memset.o \
        common/comp/mkdir.o \
        common/comp/mkfifo.o \
        common/comp/mknod.o \
        common/comp/mktemp.o \
        common/comp/mktime.o \
        common/comp/mount.o \
        common/comp/nftw.o \
        common/comp/omitted.o \
        common/comp/open.o \
        common/comp/openlog.o \
        common/comp/putenv.o \
        common/comp/re_comp.o \
        common/comp/readlink.o \
        common/comp/realpath.o \
        common/comp/regcmp.o \
        common/comp/regexp.o \
        common/comp/remove.o \
        common/comp/rename.o \
        common/comp/resolvepath.o \
        common/comp/rmdir.o \
        common/comp/setenv.o \
        common/comp/setlocale.o \
        common/comp/setlogmask.o \
        common/comp/setpgid.o \
        common/comp/setsid.o \
        common/comp/sigunblock.o \
        common/comp/sigflag.o \
        common/comp/spawnveg.o \
        common/comp/statvfs.o \
        common/comp/strcasecmp.o \
        common/comp/strchr.o \
        common/comp/strftime.o \
        common/comp/strncasecmp.o \
        common/comp/strptime.o \
```

```
127         common/comp/strrchr.o \
128         common/comp/strstr.o \
129         common/comp/strtod.o \
130         common/comp/strtol.o \
131         common/comp/strtold.o \
132         common/comp/strtoll.o \
133         common/comp/strtoul.o \
134         common/comp/strtoull.o \
135         common/comp/swab.o \
136         common/comp/symlink.o \
137         common/comp/syslog.o \
138         common/comp/system.o \
139         common/comp/tempnam.o \
140         common/comp/tmpnam.o \
141         common/comp/transition.o \
142         common/comp/tsearch.o \
143         common/comp/unlink.o \
144         common/comp/unsetenv.o \
145         common/comp/vfork.o \
146         common/comp/waitpid.o \
147         common/comp/wc.o \
148         common/comp/wordexp.o \
149         common/dir/getdents.o \
150         common/dir/opendir.o \
151         common/dir/readdir.o \
152         common/dir/rewinddir.o \
153         common/dir/seekdir.o \
154         common/dir/telldir.o \
155         common/disc/memfatal.o \
156         common/disc/sfdcdio.o \
157         common/disc/sfdcdos.o \
158         common/disc/sfdcfilter.o \
159         common/disc/sfdcmore.o \
160         common/disc/sfdcprefix.o \
161         common/disc/sfdcseekable.o \
162         common/disc/sfdcslow.o \
163         common/disc/sfdcsubstr.o \
164         common/disc/sfdctee.o \
165         common/disc/sfdcunion.o \
166         common/disc/sfkeyprintf.o \
167         common/disc/sfstrtmp.o \
168         common/hash/hashalloc.o \
169         common/hash/hashdump.o \
170         common/hash/hashfree.o \
171         common/hash/hashlast.o \
172         common/hash/hashlook.o \
173         common/hash/hashscan.o \
174         common/hash/hashsize.o \
175         common/hash/hashview.o \
176         common/hash/hashwalk.o \
177         common/hash/memhash.o \
178         common/hash/memsum.o \
179         common/hash/strhash.o \
180         common/hash/strkey.o \
181         common/hash/strsum.o \
182         common/misc/astintercept.o \
183         common/misc/debug.o \
184         common/misc/cmdarg.o \
185         common/misc/error.o \
186         common/misc/errorf.o \
187         common/misc/errormsg.o \
188         common/misc/errorx.o \
189         common/misc/fastfind.o \
190         common/misc/fmtrec.o \
191         common/misc/fs3d.o \
192         common/misc/fts.o \
```



```
193         common/misc/ftwalk.o \
194         common/misc/ftwflags.o \
195         common/misc/getcwd.o \
196         common/misc/getenv.o \
197         common/misc/glob.o \
198         common/misc/magic.o \
199         common/misc/mime.o \
200         common/misc/mimetype.o \
201         common/misc/optesc.o \
202         common/misc/optget.o \
203         common/misc/optjoin.o \
204         common/misc/optctx.o \
205         common/misc/procclose.o \
206         common/misc/procfree.o \
207         common/misc/procopen.o \
208         common/misc/procrun.o \
209         common/misc/recfmt.o \
210         common/misc/reclen.o \
211         common/misc/recstr.o \
212         common/misc/setenviron.o \
213         common/misc/sigcrit.o \
214         common/misc/sigdata.o \
215         common/misc/signal.o \
216         common/misc/stack.o \
217         common/misc/state.o \
218         common/misc/stk.o \
219         common/misc/systrace.o \
220         common/misc/translate.o \
221         common/misc/univdata.o \
222         common/obsolete/spawn.o \
223         common/path/pathaccess.o \
224         common/path/pathbin.o \
225         common/path/pathcanon.o \
226         common/path/pathcat.o \
227         common/path/pathcd.o \
228         common/path/pathcheck.o \
229         common/path/pathexists.o \
230         common/path/pathfind.o \
231         common/path/pathgetlink.o \
232         common/path/pathkey.o \
233         common/path/pathnative.o \
234         common/path/pathpath.o \
235         common/path/pathposix.o \
236         common/path/pathprobe.o \
237         common/path/pathprog.o \
238         common/path/pathrepl.o \
239         common/path/pathsetlink.o \
240         common/path/pathshell.o \
241         common/path/pathstat.o \
242         common/path/pathtemp.o \
243         common/path/pathtmp.o \
244         common/port/astconf.o \
245         common/port/astcopy.o \
246         common/port/astdynamic.o \
247         common/port/astlicense.o \
248         common/port/astquery.o \
249         common/port/aststatic.o \
250         common/port/astwinsize.o \
251         common/port/iblocks.o \
252         common/port/lc.o \
253         common/port/mc.o \
254         common/port/mnt.o \
255         common/port/touch.o \
256         common/preroot/getpreroot.o \
257         common/preroot/ispreroot.o \
258         common/preroot/realopen.o \
```

```
259         common/preroot/setpreroot.o \
260         common/regex/regalloc.o \
261         common/regex/regcache.o \
262         common/regex/regclass.o \
263         common/regex/regcoll.o \
264         common/regex/regcomp.o \
265         common/regex/regdecomp.o \
266         common/regex/regerror.o \
267         common/regex/regexec.o \
268         common/regex/regfatal.o \
269         common/regex/reginit.o \
270         common/regex/regnexec.o \
271         common/regex/regrecord.o \
272         common/regex/regrexec.o \
273         common/regex/regstat.o \
274         common/regex/regsub.o \
275         common/regex/regsubcomp.o \
276         common/regex/regsubexec.o \
277         common/sfio/_sfclrerr.o \
278         common/sfio/_sfdlen.o \
279         common/sfio/_sfeof.o \
280         common/sfio/_sferror.o \
281         common/sfio/_sffileno.o \
282         common/sfio/_sfgetc.o \
283         common/sfio/_sfgetl.o \
284         common/sfio/_sfgetl2.o \
285         common/sfio/_sfgetu.o \
286         common/sfio/_sfgetu2.o \
287         common/sfio/_sfllen.o \
288         common/sfio/_sfopen.o \
289         common/sfio/_sfputc.o \
290         common/sfio/_sfputd.o \
291         common/sfio/_sfputl.o \
292         common/sfio/_sfputm.o \
293         common/sfio/_sfputu.o \
294         common/sfio/_sfslen.o \
295         common/sfio/_sfstacked.o \
296         common/sfio/_sfulen.o \
297         common/sfio/_sfvalue.o \
298         common/sfio/sfclose.o \
299         common/sfio/sfclrlock.o \
300         common/sfio/sfcvt.o \
301         common/sfio/sfdisc.o \
302         common/sfio/sfdlen.o \
303         common/sfio/sfecvt.o \
304         common/sfio/sfexcept.o \
305         common/sfio/sfextern.o \
306         common/sfio/sffcvt.o \
307         common/sfio/sffilbuf.o \
308         common/sfio/sfflsbuf.o \
309         common/sfio/sfgetd.o \
310         common/sfio/sfgetl.o \
311         common/sfio/sfgetm.o \
312         common/sfio/sfgetr.o \
313         common/sfio/sfgetu.o \
314         common/sfio/sfllen.o \
315         common/sfio/sfmode.o \
316         common/sfio/sfmove.o \
317         common/sfio/sfmutex.o \
318         common/sfio/sfnew.o \
319         common/sfio/sfnotify.o \
320         common/sfio/sfnputc.o \
321         common/sfio/sfopen.o \
322         common/sfio/sfpeek.o \
323         common/sfio/sfpkrd.o \
324         common/sfio/sfpoll.o \
```



```
325         common/sfio/sfpool.o \
326         common/sfio/sfpopen.o \
327         common/sfio/sfprintf.o \
328         common/sfio/sfprints.o \
329         common/sfio/sfpurge.o \
330         common/sfio/sfputd.o \
331         common/sfio/sfputl.o \
332         common/sfio/sfputm.o \
333         common/sfio/sfputr.o \
334         common/sfio/sfputu.o \
335         common/sfio/sfraise.o \
336         common/sfio/sfrd.o \
337         common/sfio/sfread.o \
338         common/sfio/sfreserve.o \
339         common/sfio/sfresize.o \
340         common/sfio/sfscanf.o \
341         common/sfio/sfseek.o \
342         common/sfio/sfset.o \
343         common/sfio/sfsetbuf.o \
344         common/sfio/sfsetfd.o \
345         common/sfio/sfsize.o \
346         common/sfio/sfsk.o \
347         common/sfio/sfstack.o \
348         common/sfio/sfstrtod.o \
349         common/sfio/sfswap.o \
350         common/sfio/sfsync.o \
351         common/sfio/sftable.o \
352         common/sfio/sftell.o \
353         common/sfio/sftmp.o \
354         common/sfio/sfungetc.o \
355         common/sfio/sfvprintf.o \
356         common/sfio/sfvscanf.o \
357         common/sfio/sfwalk.o \
358         common/sfio/sfwr.o \
359         common/sfio/sfwrite.o \
360         common/stdio/_doprnt.o \
361         common/stdio/_doscan.o \
362         common/stdio/_filbuf.o \
363         common/stdio/_flsbuf.o \
364         common/stdio/_stdfun.o \
365         common/stdio/_stdopen.o \
366         common/stdio/_stdprintf.o \
367         common/stdio/_stdscanf.o \
368         common/stdio/_stdsprnt.o \
369         common/stdio/_stdvbuf.o \
370         common/stdio/_stdvsnprnt.o \
371         common/stdio/_stdvsprnt.o \
372         common/stdio/_stdvsscn.o \
373         common/stdio/asprintf.o \
374         common/stdio/clearerr.o \
375         common/stdio/fclose.o \
376         common/stdio/fcloseall.o \
377         common/stdio/fdopen.o \
378         common/stdio/feof.o \
379         common/stdio/ferror.o \
380         common/stdio/fflush.o \
381         common/stdio/fgetc.o \
382         common/stdio/fgetpos.o \
383         common/stdio/fgets.o \
384         common/stdio/fgetwc.o \
385         common/stdio/fgetws.o \
386         common/stdio/fileno.o \
387         common/stdio/flockfile.o \
388         common/stdio/fmemopen.o \
389         common/stdio/fopen.o \
390         common/stdio/fprintf.o \
```

```
391         common/stdio/fpurge.o \
392         common/stdio/fputc.o \
393         common/stdio/fputs.o \
394         common/stdio/fputwc.o \
395         common/stdio/fputws.o \
396         common/stdio/funlockfile.o \
397         common/stdio/fread.o \
398         common/stdio/freopen.o \
399         common/stdio/fscanf.o \
400         common/stdio/fseek.o \
401         common/stdio/fseeko.o \
402         common/stdio/fsetpos.o \
403         common/stdio/ftell.o \
404         common/stdio/ftello.o \
405         common/stdio/ftrylockfile.o \
406         common/stdio/fwide.o \
407         common/stdio/fwprintf.o \
408         common/stdio/fwrite.o \
409         common/stdio/fwscanf.o \
410         common/stdio/getc.o \
411         common/stdio/getchar.o \
412         common/stdio/getdelim.o \
413         common/stdio/getline.o \
414         common/stdio/getw.o \
415         common/stdio/getwc.o \
416         common/stdio/getwchar.o \
417         common/stdio/pclose.o \
418         common/stdio/popen.o \
419         common/stdio/printf.o \
420         common/stdio/putc.o \
421         common/stdio/putchar.o \
422         common/stdio/puts.o \
423         common/stdio/putw.o \
424         common/stdio/putwc.o \
425         common/stdio/putwchar.o \
426         common/stdio/rewind.o \
427         common/stdio/scanf.o \
428         common/stdio/setbuf.o \
429         common/stdio/setbuffer.o \
430         common/stdio/setlinebuf.o \
431         common/stdio/setvbuf.o \
432         common/stdio/snprintf.o \
433         common/stdio/sprintf.o \
434         common/stdio/sscanf.o \
435         common/stdio/stdio_c99.o \
436         common/stdio/swprintf.o \
437         common/stdio/swscanf.o \
438         common/stdio/tmpfile.o \
439         common/stdio/ungetc.o \
440         common/stdio/ungetwc.o \
441         common/stdio/vasprintf.o \
442         common/stdio/vfprintf.o \
443         common/stdio/vfscanf.o \
444         common/stdio/vfwprintf.o \
445         common/stdio/vfwscanf.o \
446         common/stdio/vprintf.o \
447         common/stdio/vscanf.o \
448         common/stdio/vsnprintf.o \
449         common/stdio/vsprintf.o \
450         common/stdio/vsscanf.o \
451         common/stdio/vswprintf.o \
452         common/stdio/vswscanf.o \
453         common/stdio/vwprintf.o \
454         common/stdio/vwscanf.o \
455         common/stdio/wprintf.o \
456         common/stdio/wscanf.o \
```



```
457         common/string/base64.o \
458         common/string/ccmap.o \
459         common/string/ccmapid.o \
460         common/string/ccnative.o \
461         common/string/chresc.o \
462         common/string/chrtoi.o \
463         common/string/fmtbase.o \
464         common/string/fmtbuf.o \
465         common/string/fmtclock.o \
466         common/string/fmtdev.o \
467         common/string/fmtelapsed.o \
468         common/string/fmterror.o \
469         common/string/fmtesc.o \
470         common/string/fmtfmt.o \
471         common/string/fmtfs.o \
472         common/string/fmtgid.o \
473         common/string/fmtident.o \
474         common/string/fmtip4.o \
475         common/string/fmtip6.o \
476         common/string/fmtls.o \
477         common/string/fmtmatch.o \
478         common/string/fmtmode.o \
479         common/string/fmtnum.o \
480         common/string/fmtperm.o \
481         common/string/fmtre.o \
482         common/string/fmtscale.o \
483         common/string/fmtsignal.o \
484         common/string/fmttime.o \
485         common/string/fmttmx.o \
486         common/string/fmttv.o \
487         common/string/fmtuid.o \
488         common/string/fmtversion.o \
489         common/string/memdup.o \
490         common/string/modedata.o \
491         common/string/modei.o \
492         common/string/modex.o \
493         common/string/stracmp.o \
494         common/string/strcopy.o \
495         common/string/strdup.o \
496         common/string/strelapsed.o \
497         common/string/strerror.o \
498         common/string/stresc.o \
499         common/string/streval.o \
500         common/string/strexpr.o \
501         common/string/strgid.o \
502         common/string/strlcat.o \
503         common/string/strlcpy.o \
504         common/string/strlook.o \
505         common/string/strmatch.o \
506         common/string/strmode.o \
507         common/string/strnacmp.o \
508         common/string/strncopy.o \
509         common/string/strnpcmp.o \
510         common/string/strntod.o \
511         common/string/strntol.o \
512         common/string/strntold.o \
513         common/string/strntoll.o \
514         common/string/strnton.o \
515         common/string/strntoul.o \
516         common/string/strntonll.o \
517         common/string/strntoull.o \
518         common/string/strnvcmp.o \
519         common/string/stropt.o \
520         common/string/strpcmp.o \
521         common/string/strperm.o \
522         common/string/strpsearch.o \
```

```
523         common/string/strsearch.o \
524         common/string/strsort.o \
525         common/string/strtape.o \
526         common/string/strtoip4.o \
527         common/string/strtoip6.o \
528         common/string/strton.o \
529         common/string/strtonll.o \
530         common/string/struid.o \
531         common/string/struniq.o \
532         common/string/strvcmp.o \
533         common/string/swapget.o \
534         common/string/swapmem.o \
535         common/string/swapop.o \
536         common/string/swapput.o \
537         common/string/tok.o \
538         common/string/tokline.o \
539         common/string/tokscan.o \
540         common/tm/tmdata.o \
541         common/tm/tmdate.o \
542         common/tm/tmequiv.o \
543         common/tm/tmfix.o \
544         common/tm/tmfmt.o \
545         common/tm/tmform.o \
546         common/tm/tmgoff.o \
547         common/tm/tminit.o \
548         common/tm/tmleap.o \
549         common/tm/tmlex.o \
550         common/tm/tmlocale.o \
551         common/tm/tmmake.o \
552         common/tm/tmpoff.o \
553         common/tm/tmscan.o \
554         common/tm/tmsleep.o \
555         common/tm/tmtime.o \
556         common/tm/tmtype.o \
557         common/tm/tmweek.o \
558         common/tm/tmword.o \
559         common/tm/tmxdate.o \
560         common/tm/tmxduration.o \
561         common/tm/tmxfmt.o \
562         common/tm/tmxgettime.o \
563         common/tm/tmxleap.o \
564         common/tm/tmxmake.o \
565         common/tm/tmxscan.o \
566         common/tm/tmxsettime.o \
567         common/tm/tmxsleep.o \
568         common/tm/tmxtime.o \
569         common/tm/tmxtouch.o \
570         common/tm/tmzone.o \
571         common/tm/tvcmp.o \
572         common/tm/tvgettime.o \
573         common/tm/tvsettime.o \
574         common/tm/tvsleep.o \
575         common/tm/tvtouch.o \
576         common/uwin/a64l.o \
577         common/uwin/acosh.o \
578         common/uwin/asinh.o \
579         common/uwin/atanh.o \
580         common/uwin/cbrt.o \
581         common/uwin/crypt.o \
582         common/uwin/erf.o \
583         common/uwin/err.o \
584         common/uwin/exp.o \
585         common/uwin/exp__E.o \
586         common/uwin/expm1.o \
587         common/uwin/gamma.o \
588         common/uwin/getpass.o \
```



```
589         common/uwin/lgamma.o \
590         common/uwin/log.o \
591         common/uwin/log1p.o \
592         common/uwin/log__L.o \
593         common/uwin/rand48.o \
594         common/uwin/random.o \
595         common/uwin/rcmd.o \
596         common/uwin/rint.o \
597         common/uwin/support.o \
598         common/vec/vecargs.o \
599         common/vec/vecfile.o \
600         common/vec/vecfree.o \
601         common/vec/vecload.o \
602         common/vec/vecstring.o \
603         common/vmalloc/malloc.o \
604         common/vmalloc/vmbest.o \
605         common/vmalloc/vmclear.o \
606         common/vmalloc/vmclose.o \
607         common/vmalloc/vmdcheap.o \
608         common/vmalloc/vmdebug.o \
609         common/vmalloc/vmdisc.o \
610         common/vmalloc/vmexit.o \
611         common/vmalloc/vmgetmem.o \
612         common/vmalloc/vmlast.o \
613         common/vmalloc/vmmopen.o \
614         common/vmalloc/vmopen.o \
615         common/vmalloc/vmpool.o \
616         common/vmalloc/vmprivate.o \
617         common/vmalloc/vmprofile.o \
618         common/vmalloc/vmregion.o \
619         common/vmalloc/vmsegment.o \
620         common/vmalloc/vmset.o \
621         common/vmalloc/vmstat.o \
622         common/vmalloc/vmstrdup.o \
623         common/vmalloc/vmtrace.o \
624         common/vmalloc/vmwalk.o

626 # We are storing the object files into subdirs avoid the
627 # confusion with having 550+ object files in the toplevel pics/
628 # directory (this matches the way how the original AST build system
629 # deals with this "logistic" issue) - the rules below ensure that
630 # the destination directory is available.
631 OBJDIRS += \
632         common/cdt \
633         common/comp \
634         common/dir \
635         common/disc \
636         common/hash \
637         common/misc \
638         common/obsolete \
639         common/path \
640         common/port \
641         common/preroot \
642         common/regex \
643         common/sfio \
644         common/stdio \
645         common/string \
646         common/tm \
647         common/uwin \
648         common/vec \
649         common/vmalloc
650 PICSDIRS= $(OBJDIRS:%=pics/%)
651 mkpicdirs:
652         @mkdir -p $(PICSDIRS)

654 # We need our own rules here since some source files come from
```

```
655 # the plaftorm-specific directories and the default rules do
656 # not cover this
657 pics/%.o: ../%.c
658         $(CC) $(CFLAGS) $(CPPFLAGS) -c -o $@ $<
659         $(POST_PROCESS_O)

661 include ../../Makefile.astmsg

663 include ../../Makefile.lib

665 # mapfile-vers does not live with the sources in in common/ to make
666 # automated code updates easier.
667 MAPFILES=       ../mapfile-vers

669 # Set common AST build flags (e.g. C99/XPG6, needed to support the math stuff)
670 include ../../../Makefile.ast

672 # special rule because sources live both ../common (normal)
673 # and $(TRANSMACH) (generated)
674 SRCS=           $(OBJECTS:%.o=../%.c)

676 LIBS =          $(DYNLIB) $(LINTLIB)

678 LDLIBS += \
679         -lsocket \
680         -lm \
681         -lc

683 $(LINTLIB) :=   SRCS = $(SRCDIR)/$(LINTSRC)

685 SRCDIR =        ../common

687 # We use "=" here since using $(CPPFLAGS.master) is very tricky in our
688 # case - it MUST come as the last element but future changes in -D options
689 # may then cause silent breakage in the AST sources because the last -D
690 # option specified overrides previous -D options so we prefer the current
691 # way to explicitly list each single flag.
692 # Notes:
693 #   - "-D_BLD_DLL" comes from ${mam_cc_DLL} in Mamfile
694 #   - Be careful with "-D__OBSOLETE__=xxx". Make sure this is in sync with
695 #     upstream (see Mamfile) and do not change the |__OBSOLETE__| value
696 #     without examining the symbols that will be removed, and evaluating
697 #     whether that breaks compatibility with upstream binaries.
698 CPPFLAGS = \
699         $(DTEXTDOM) $(DTS_ERRNO) \
700         $(ASTPLATFORMCPPFLAGS) \
701         -Isrc/lib/libast \
702         -I$(SRCDIR) \
703         -I$(SRCDIR)/comp \
704         -I$(SRCDIR)/include \
705         -I$(SRCDIR)/std \
706         -I$(SRCDIR)/dir \
707         -I$(SRCDIR)/port \
708         -I$(SRCDIR)/sfio \
709         -I$(SRCDIR)/astsa \
710         -I$(SRCDIR)/misc \
711         -I$(SRCDIR)/string \
712         -Iinclude/ast \
713         -I$(ROOT)/usr/include \
714         '-DCONF_LIBSUFFIX=".so"' \
715         '-DCONF_LIBPREFIX="lib"' \
716         -DERROR_CATALOG=\""libast"\" \
717         -D__OBSOLETE__=20100101 \
718         -D_BLD_ast \
719         -D_PACKAGE_ast \
720         -D_BLD_DLL
```



```
722 CFLAGS += \
723         $(ASTCFLAGS)
724 CFLAGS64 += \
725         $(ASTCFLAGS64)

727 CERRWARN += -_gcc=-Wno-parentheses
728 CERRWARN += -_gcc=-Wno-uninitialized
729 CERRWARN += -_gcc=-Wno-char-subscripts
730 CERRWARN += -_gcc=-Wno-clobbered
731 CERRWARN += -_gcc=-Wno-unused-variable
732 CERRWARN += -_gcc=-Wno-unused-but-set-variable
733 CERRWARN += -_gcc=-Wno-unused-but-set-parameter
734 CERRWARN += -_gcc=-Wno-unused-value
735 CERRWARN += -_gcc=-Wno-unused-function
736 CERRWARN += -_gcc=-Wno-unused-label
737 CERRWARN += -_gcc=-Wno-implicit-function-declaration
738 CERRWARN += -_gcc=-Wno-empty-body
739 CERRWARN += -_gcc=-Wno-type-limits
740 CERRWARN += -_gcc=-Wno-address

742 SMATCH=off

744 pics/$(MACH)/src/lib/libast/conftab.o \
745 pics/$(MACH64)/src/lib/libast/conftab.o := CERRWARN += -erroff=E_INIT_DOES_NOT_F
746 pics/common/comp/setlocale.o            := CERRWARN += -erroff=E_INTEGER_OVERFLO
747 pics/common/comp/setlocale.o            := CERRWARN += -erroff=E_INIT_DOES_NOT_F
748 pics/common/comp/setlocale.o            := CERRWARN += -erroff=E_INIT_SIGN_EXTEN
749 pics/common/hash/hashlook.o             := CERRWARN += -erroff=E_CONST_PROMOTED_
750 pics/common/hash/memhash.o              := CERRWARN += -erroff=E_CONST_PROMOTED_
751 pics/common/hash/memsum.o               := CERRWARN += -erroff=E_CONST_PROMOTED_
752 pics/common/hash/strhash.o              := CERRWARN += -erroff=E_CONST_PROMOTED_
753 pics/common/hash/strsum.o               := CERRWARN += -erroff=E_CONST_PROMOTED_
754 pics/common/misc/recstr.o               := CERRWARN += -erroff=E_INTEGER_OVERFLO
755 pics/common/misc/translate.o            := CERRWARN += -erroff=E_INTEGER_OVERFLO
756 pics/common/path/pathkey.o              := CERRWARN += -erroff=E_CONST_PROMOTED_
757 pics/common/port/astconf.o              := CERRWARN += -erroff=E_CONST_OBJ_SHOUL
758 pics/common/stdio/fflush.o              := CERRWARN += -erroff=E_NO_IMPLICIT_DEC
759 pics/common/stdio/getline.o             := CERRWARN += -erroff=E_NO_IMPLICIT_DEC
760 pics/common/sfio/sfmove.o               := CERRWARN += -erroff=E_NO_IMPLICIT_DEC
761 pics/common/sfio/sfrd.o                 := CERRWARN += -erroff=E_NO_IMPLICIT_DEC
762 pics/common/sfio/sfvscanf.o             := CERRWARN += -erroff=E_END_OF_LOOP_COD
763 pics/common/tm/tmxduration.o            := CERRWARN += -erroff=E_NO_IMPLICIT_DEC

765 .KEEP_STATE:

767 all: mkpicdirs .WAIT $(LIBS)

769 #
770 # libast is not lint-clean yet; fake up a target.  (You can use
771 # "make lintcheck" to actually run lint; please send all lint fixes
772 # upstream (to AT&T) so the next update will pull them into ON.)
773 #
774 lint:
775         @ print "usr/src/lib/libast is not lint-clean: skipping"

777 include ../../Makefile.targ
```

```
**********************************************************
   23545 Fri Dec 21 14:59:56 2018
new/usr/src/lib/libc/amd64/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2016 Joyent, Inc.
# Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2013 Garrett D'Amore <garrett@damore.org>
# Copyright 2018 Nexenta Systems, Inc.
# Copyright (c) 2018, Joyent, Inc.
#

LIBCBASE=       .
LIBCDIR=        $(SRC)/lib/libc
LIBRARY=        libc.a
LIB_PIC=        libc_pic.a
VERS=           .1
CPP=            /usr/lib/cpp
TARGET_ARCH=    amd64

# include comm page definitions
include $(SRC)/lib/commpage/Makefile.shared.com
include $(SRC)/lib/commpage/Makefile.shared.targ

# objects are grouped by source directory

# local objects
STRETS=

CRTOBJS=                        \
        cerror.o

DYNOBJS=

FPOBJS=                         \
        fpgetmask.o             \
        fpgetround.o            \
        fpsetmask.o             \
        fpsetround.o            \
        fpstart.o               \
        ieee.o
```



```
I386FPOBJS=                     \
        _D_cplx_div.o           \
        _D_cplx_div_ix.o        \
        _D_cplx_div_rx.o        \
        _D_cplx_lr_div.o        \
        _D_cplx_lr_div_ix.o     \
        _D_cplx_lr_div_rx.o     \
        _D_cplx_mul.o           \
        _F_cplx_div.o           \
        _F_cplx_div_ix.o        \
        _F_cplx_div_rx.o        \
        _F_cplx_lr_div.o        \
        _F_cplx_lr_div_ix.o     \
        _F_cplx_lr_div_rx.o     \
        _F_cplx_mul.o           \
        _X_cplx_div.o           \
        _X_cplx_div_ix.o        \
        _X_cplx_div_rx.o        \
        _X_cplx_lr_div.o        \
        _X_cplx_lr_div_ix.o     \
        _X_cplx_lr_div_rx.o     \
        _X_cplx_mul.o

FPASMOBJS=                      \
        __xgetRD.o              \
        _xtoll.o                \
        _xtoull.o               \
        _base_il.o              \
        fpcw.o                  \
        fpgetsticky.o           \
        fpsetsticky.o

ATOMICOBJS=                     \
        atomic.o

CHACHAOBJS=                     \
        chacha.o

XATTROBJS=                      \
        xattr_common.o
COMOBJS=                        \
        bcmp.o                  \
        bcopy.o                 \
        bsearch.o               \
        bzero.o                 \
        qsort.o                 \
        strtol.o                \
        strtoul.o               \
        strtoll.o               \
        strtoull.o

GENOBJS=                        \
        $(COMMPAGE_OBJS)        \
        _getsp.o                \
        abs.o                   \
        alloca.o                \
        arc4random.o            \
        arc4random_uniform.o    \
        attrat.o                \
        byteorder.o             \
        cuexit.o                \
        ecvt.o                  \
        endian.o                \
        errlst.o                \
        amd64_data.o            \
        ldivide.o               \
```

```
127         lock.o                  \
128         makectxt.o              \
129         memccpy.o               \
130         memchr.o                \
131         memcmp.o                \
132         memcpy.o                \
133         memset.o                \
134         new_list.o              \
135         proc64_id.o             \
136         proc64_support.o        \
137         setjmp.o                \
138         siginfolst.o            \
139         siglongjmp.o            \
140         strcmp.o                \
141         strcpy.o                \
142         strlen.o                \
143         strncmp.o               \
144         strncpy.o               \
145         strnlen.o               \
146         sync_instruction_memory.o

148 # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
149 # This macro should ALWAYS be empty; native APIs are already 'large file'.
150 COMSYSOBJS64=

152 SYSOBJS64=

154 COMSYSOBJS=                     \
155         __clock_timer.o         \
156         __getloadavg.o          \
157         __rusagesys.o           \
158         __signotify.o           \
159         __sigrt.o               \
160         __time.o                \
161         _lgrp_home_fast.o       \
162         _lgrpsys.o              \
163         _nfssys.o               \
164         _portfs.o               \
165         _pset.o                 \
166         _rpcsys.o               \
167         _sigaction.o            \
168         _so_accept.o            \
169         _so_bind.o              \
170         _so_connect.o           \
171         _so_getpeername.o       \
172         _so_getsockname.o       \
173         _so_getsockopt.o        \
174         _so_listen.o            \
175         _so_recv.o              \
176         _so_recvfrom.o          \
177         _so_recvmsg.o           \
178         _so_send.o              \
179         _so_sendmsg.o           \
180         _so_sendto.o            \
181         _so_setsockopt.o        \
182         _so_shutdown.o          \
183         _so_socket.o            \
184         _so_socketpair.o        \
185         _sockconfig.o           \
186         acct.o                  \
187         acl.o                   \
188         adjtime.o               \
189         alarm.o                 \
190         brk.o                   \
191         chdir.o                 \
192         chroot.o                \
```



```
193         cladm.o                 \
194         close.o                 \
195         execve.o                \
196         exit.o                  \
197         facl.o                  \
198         fchdir.o                \
199         fchroot.o               \
200         fdsync.o                \
201         fpathconf.o             \
202         fstatfs.o               \
203         fstatvfs.o              \
204         getcpuid.o              \
205         getdents.o              \
206         getegid.o               \
207         geteuid.o               \
208         getgid.o                \
209         getgroups.o             \
210         gethrtime.o             \
211         getitimer.o             \
212         getmsg.o                \
213         getpid.o                \
214         getpmsg.o               \
215         getppid.o               \
216         getrandom.o             \
217         getrlimit.o             \
218         getuid.o                \
219         gtty.o                  \
220         install_utrap.o         \
221         ioctl.o                 \
222         kaio.o                  \
223         kill.o                  \
224         llseek.o                \
225         lseek.o                 \
226         mmapobjsys.o            \
227         memcntl.o               \
228         mincore.o               \
229         mmap.o                  \
230         modctl.o                \
231         mount.o                 \
232         mprotect.o              \
233         munmap.o                \
234         nice.o                  \
235         ntp_adjtime.o           \
236         ntp_gettime.o           \
237         p_online.o              \
238         pathconf.o              \
239         pause.o                 \
240         pcsample.o              \
241         pipe2.o                 \
242         pollsys.o               \
243         pread.o                 \
244         preadv.o                \
245         priocntlset.o           \
246         processor_bind.o        \
247         processor_info.o        \
248         profil.o                \
249         psecflagsset.o          \
250         putmsg.o                \
251         putpmsg.o               \
252         pwrite.o                \
253         pwritev.o               \
254         read.o                  \
255         readv.o                 \
256         resolvepath.o           \
257         seteguid.o              \
258         setgid.o                \
```

```
259         setgroups.o             \
260         setitimer.o             \
261         setreid.o               \
262         setrlimit.o             \
263         setuid.o                \
264         sigaltstk.o             \
265         sigprocmsk.o            \
266         sigsendset.o            \
267         sigsuspend.o            \
268         statfs.o                \
269         statvfs.o               \
270         stty.o                  \
271         sync.o                  \
272         sysconfig.o             \
273         sysfs.o                 \
274         sysinfo.o               \
275         syslwp.o                \
276         times.o                 \
277         ulimit.o                \
278         umask.o                 \
279         umount2.o               \
280         utssys.o                \
281         uucopy.o                \
282         vhangup.o               \
283         waitid.o                \
284         write.o                 \
285         writev.o                \
286         yield.o

288 SYSOBJS=                        \
289         __clock_gettime.o       \
290         __clock_gettime_sys.o   \
291         __getcontext.o          \
292         __uadmin.o              \
293         _lwp_mutex_unlock.o     \
294         _stack_grow.o           \
295         door.o                  \
296         forkx.o                 \
297         forkallx.o              \
298         getcontext.o            \
299         gettimeofday.o          \
300         lwp_private.o           \
301         nuname.o                \
302         syscall.o               \
303         sysi86.o                \
304         tls_get_addr.o          \
305         uadmin.o                \
306         umount.o                \
307         uname.o                 \
308         vforkx.o

310 # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
311 # This macro should ALWAYS be empty; native APIs are already 'large file'.
312 PORTGEN64=

314 # objects from source under $(LIBCDIR)/port
315 PORTFP=                         \
316         __flt_decim.o           \
317         __flt_rounds.o          \
318         __tbl_10_b.o            \
319         __tbl_10_h.o            \
320         __tbl_10_s.o            \
321         __tbl_2_b.o             \
322         __tbl_2_h.o             \
323         __tbl_2_s.o             \
324         __tbl_fdq.o             \
```



```
325         __tbl_tens.o            \
326         __x_power.o             \
327         _base_sup.o             \
328         aconvert.o              \
329         decimal_bin.o           \
330         double_decim.o          \
331         econvert.o              \
332         fconvert.o              \
333         file_decim.o            \
334         finite.o                \
335         fp_data.o               \
336         func_decim.o            \
337         gconvert.o              \
338         hex_bin.o               \
339         ieee_globals.o          \
340         pack_float.o            \
341         sigfpe.o                \
342         string_decim.o

344 PORTGEN=                        \
345         _env_data.o             \
346         _xftw.o                 \
347         a64l.o                  \
348         abort.o                 \
349         addsev.o                \
350         ascii_strcasecmp.o      \
351         ascii_strncasecmp.o     \
352         assert.o                \
353         atexit.o                \
354         atfork.o                \
355         atof.o                  \
356         atoi.o                  \
357         atol.o                  \
358         atoll.o                 \
359         attropen.o              \
360         basename.o              \
361         calloc.o                \
362         catgets.o               \
363         catopen.o               \
364         cfgetispeed.o           \
365         cfgetospeed.o           \
366         cfree.o                 \
367         cfsetispeed.o           \
368         cfsetospeed.o           \
369         cftime.o                \
370         clock.o                 \
371         closedir.o              \
372         closefrom.o             \
373         confstr.o               \
374         crypt.o                 \
375         csetlen.o               \
376         ctime.o                 \
377         ctime_r.o               \
378         daemon.o                \
379         deflt.o                 \
380         directio.o              \
381         dirname.o               \
382         div.o                   \
383         drand48.o               \
384         dup.o                   \
385         env_data.o              \
386         err.o                   \
387         errno.o                 \
388         euclen.o                \
389         event_port.o            \
390         execvp.o                \
```

```
391         explicit_bzero.o        \
392         fattach.o               \
393         fdetach.o               \
394         fdopendir.o             \
395         ffs.o                   \
396         flock.o                 \
397         fls.o                   \
398         fmtmsg.o                \
399         freezero.o              \
400         ftime.o                 \
401         ftok.o                  \
402         fts.o                   \
403         ftw.o                   \
404         gcvt.o                  \
405         get_nprocs.o            \
406         getauxv.o               \
407         getcwd.o                \
408         getdate_err.o           \
409         getdtblsize.o           \
410         getentropy.o            \
411         getenv.o                \
412         getexecname.o           \
413         getgrnam.o              \
414         getgrnam_r.o            \
415         gethostid.o             \
416         gethostname.o           \
417         gethz.o                 \
418         getisax.o               \
419         getloadavg.o            \
420         getlogin.o              \
421         getmntent.o             \
422         getnetgrent.o           \
423         getopt.o                \
424         getopt_long.o           \
425         getpagesize.o           \
426         getpw.o                 \
427         getpwnam.o              \
428         getpwnam_r.o            \
429         getrusage.o             \
430         getspent.o              \
431         getspent_r.o            \
432         getsubopt.o             \
433         gettxt.o                \
434         getusershell.o          \
435         getut.o                 \
436         getutx.o                \
437         getvfsent.o             \
438         getwd.o                 \
439         getwidth.o              \
440         getxby_door.o           \
441         gtxt.o                  \
442         hsearch.o               \
443         iconv.o                 \
444         imaxabs.o               \
445         imaxdiv.o               \
446         index.o                 \
447         initgroups.o            \
448         insque.o                \
449         isaexec.o               \
450         isastream.o             \
451         isatty.o                \
452         killpg.o                \
453         klpdlib.o               \
454         l64a.o                  \
455         lckpwdf.o               \
456         lconstants.o            \
```



```
457         lexp10.o                \
458         lfind.o                 \
459         lfmt.o                  \
460         lfmt_log.o              \
461         lldiv.o                 \
462         llog10.o                \
463         lltostr.o               \
464         lmath.o                 \
465         localtime.o             \
466         lsearch.o               \
467         madvise.o               \
468         malloc.o                \
469         memalign.o              \
470         memmem.o                \
471         memset_s.o              \
472         mkdev.o                 \
473         mkdtemp.o               \
474         mkfifo.o                \
475         mkstemp.o               \
476         mktemp.o                \
477         mlock.o                 \
478         mlockall.o              \
479         mon.o                   \
480         msync.o                 \
481         munlock.o               \
482         munlockall.o            \
483         ndbm.o                  \
484         nftw.o                  \
485         nlspath_checks.o        \
486         nsparse.o               \
487         nss_common.o            \
488         nss_dbdefs.o            \
489         nss_deffinder.o         \
490         opendir.o               \
491         opt_data.o              \
492         perror.o                \
493         pfmt.o                  \
494         pfmt_data.o             \
495         pfmt_print.o            \
496         pipe.o                  \
497         plock.o                 \
498         poll.o                  \
499         posix_fadvise.o         \
500         posix_fallocate.o       \
501         posix_madvise.o         \
502         posix_memalign.o        \
503         priocntl.o              \
504         priv_str_xlate.o        \
505         privlib.o               \
506         psecflags.o             \
507         psiginfo.o              \
508         psignal.o               \
509         pt.o                    \
510         putpwent.o              \
511         putspent.o              \
512         raise.o                 \
513         rand.o                  \
514         random.o                \
515         rctlops.o               \
516         readdir.o               \
517         readdir_r.o             \
518         reallocarray.o          \
519         recallocarray.o         \
520         realpath.o              \
521         reboot.o                \
522         regexpr.o               \
```

```
523         remove.o                \
524         rewinddir.o             \
525         rindex.o                \
526         scandir.o               \
527         seekdir.o               \
528         select.o                \
529         set_constraint_handler_s.o \
530         setlabel.o              \
531         setpriority.o           \
532         settimeofday.o          \
533         sh_locks.o              \
534         sigflag.o               \
535         siglist.o               \
536         sigsend.o               \
537         sigsetops.o             \
538         ssignal.o               \
539         stack.o                 \
540         stpcpy.o                \
541         stpncpy.o               \
542         str2sig.o               \
543         strcase_charmap.o       \
544         strcat.o                \
545         strchr.o                \
546         strchrnul.o             \
547         strcspn.o               \
548         strdup.o                \
549         strerror.o              \
550         strlcat.o               \
551         strlcpy.o               \
552         strncat.o               \
553         strndup.o               \
554         strpbrk.o               \
555         strrchr.o               \
556         strsep.o                \
557         strsignal.o             \
558         strspn.o                \
559         strstr.o                \
560         strtod.o                \
561         strtoimax.o             \
562         strtok.o                \
563         strtok_r.o              \
564         strtonum.o              \
565         strtoumax.o             \
566         swab.o                  \
567         swapctl.o               \
568         sysconf.o               \
569         syslog.o                \
570         tcdrain.o               \
571         tcflow.o                \
572         tcflush.o               \
573         tcgetattr.o             \
574         tcgetpgrp.o             \
575         tcgetsid.o              \
576         tcsendbreak.o           \
577         tcsetattr.o             \
578         tcsetpgrp.o             \
579         tell.o                  \
580         telldir.o               \
581         tfind.o                 \
582         time_data.o             \
583         time_gdata.o            \
584         timespec_get.o          \
585         tls_data.o              \
586         truncate.o              \
587         tsdalloc.o              \
588         tsearch.o               \
```



```
589         ttyname.o               \
590         ttyslot.o               \
591         ualarm.o                \
592         ucred.o                 \
593         valloc.o                \
594         vlfmt.o                 \
595         vpfmt.o                 \
596         waitpid.o               \
597         walkstack.o             \
598         wdata.o                 \
599         xgetwidth.o             \
600         xpg4.o                  \
601         xpg6.o

603 PORTINET=                       \
604         inet_lnaof.o            \
605         inet_makeaddr.o         \
606         inet_network.o          \
607         inet_ntoa.o             \
608         inet_ntop.o             \
609         inet_pton.o

611 PORTPRINT_W=                    \
612         doprnt_w.o

614 PORTPRINT=                     \
615         asprintf.o              \
616         doprnt.o                \
617         fprintf.o               \
618         printf.o                \
619         snprintf.o              \
620         sprintf.o               \
621         vfprintf.o              \
622         vprintf.o               \
623         vsnprintf.o             \
624         vsprintf.o              \
625         vwprintf.o              \
626         wprintf.o

628 # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
629 # This macro should ALWAYS be empty; native APIs are already 'large file'.
630 PORTSTDIO64=

632 PORTSTDIO_W=                    \
633         doscan_w.o

635 PORTSTDIO=                     \
636         __extensions.o          \
637         _endopen.o              \
638         _filbuf.o               \
639         _findbuf.o              \
640         _flsbuf.o               \
641         _wrtchk.o               \
642         clearerr.o              \
643         ctermid.o               \
644         ctermid_r.o             \
645         cuserid.o               \
646         data.o                  \
647         doscan.o                \
648         fdopen.o                \
649         feof.o                  \
650         ferror.o                \
651         fgetc.o                 \
652         fgets.o                 \
653         fileno.o                \
654         flockf.o                \
```

```
655         flush.o                 \
656         fopen.o                 \
657         fpos.o                  \
658         fputc.o                 \
659         fputs.o                 \
660         fread.o                 \
661         fseek.o                 \
662         fseeko.o                \
663         ftell.o                 \
664         ftello.o                \
665         fwrite.o                \
666         getc.o                  \
667         getchar.o               \
668         getline.o               \
669         getpass.o               \
670         gets.o                  \
671         getw.o                  \
672         mse.o                   \
673         popen.o                 \
674         putc.o                  \
675         putchar.o               \
676         puts.o                  \
677         putw.o                  \
678         rewind.o                \
679         scanf.o                 \
680         setbuf.o                \
681         setbuffer.o             \
682         setvbuf.o               \
683         system.o                \
684         tempnam.o               \
685         tmpfile.o               \
686         tmpnam_r.o              \
687         ungetc.o                \
688         vscanf.o                \
689         vwscanf.o               \
690         wscanf.o

692 PORTI18N=                       \
693         getwchar.o              \
694         putwchar.o              \
695         putws.o                 \
696         strtows.o               \
697         wcsnlen.o               \
698         wcsstr.o                \
699         wcstoimax.o             \
700         wcstol.o                \
701         wcstoul.o               \
702         wcswcs.o                \
703         wmemchr.o               \
704         wmemcmp.o               \
705         wmemcpy.o               \
706         wmemmove.o              \
707         wmemset.o               \
708         wscat.o                 \
709         wschr.o                 \
710         wscmp.o                 \
711         wscpy.o                 \
712         wscspn.o                \
713         wsdup.o                 \
714         wslen.o                 \
715         wsncat.o                \
716         wsncmp.o                \
717         wsncpy.o                \
718         wspbrk.o                \
719         wsprintf.o              \
720         wsrchr.o                \
```



```
721         wsscanf.o               \
722         wsspn.o                 \
723         wstod.o                 \
724         wstok.o                 \
725         wstol.o                 \
726         wstoll.o                \
727         wsxfrm.o                \
728         gettext.o               \
729         gettext_gnu.o           \
730         gettext_real.o          \
731         gettext_util.o          \
732         plural_parser.o         \
733         wdresolve.o             \
734         _ctype.o                \
735         isascii.o               \
736         toascii.o

738 PORTI18N_COND=                  \
739         wcstol_longlong.o       \
740         wcstoul_longlong.o

742 PORTLOCALE=                    \
743         big5.o                  \
744         btowc.o                 \
745         collate.o               \
746         collcmp.o               \
747         euc.o                   \
748         fnmatch.o               \
749         fgetwc.o                \
750         fgetws.o                \
751         fix_grouping.o          \
752         fputwc.o                \
753         fputws.o                \
754         fwide.o                 \
755         gb18030.o               \
756         gb2312.o                \
757         gbk.o                   \
758         getdate.o               \
759         isdigit.o               \
760         iswctype.o              \
761         ldpart.o                \
762         lmessages.o             \
763         lnumeric.o              \
764         lmonetary.o             \
765         localeconv.o            \
766         localeimpl.o            \
767         mbftowc.o               \
768         mblen.o                 \
769         mbrlen.o                \
770         mbrtowc.o               \
771         mbsinit.o               \
772         mbsnrtowcs.o            \
773         mbsrtowcs.o             \
774         mbstowcs.o              \
775         mbtowc.o                \
776         mskanji.o               \
777         nextwctype.o            \
778         nl_langinfo.o           \
779         none.o                  \
780         rune.o                  \
781         runetype.o              \
782         setlocale.o             \
783         setrunelocale.o         \
784         strcasecmp.o            \
785         strcasestr.o            \
786         strcoll.o               \
```

```
787         strfmon.o               \
788         strftime.o              \
789         strncasecmp.o           \
790         strptime.o              \
791         strxfrm.o               \
792         table.o                 \
793         timelocal.o             \
794         tolower.o               \
795         towlower.o              \
796         ungetwc.o               \
797         utf8.o                  \
798         wcrtomb.o               \
799         wcscasecmp.o            \
800         wcscoll.o               \
801         wcsftime.o              \
802         wcsnrtombs.o            \
803         wcsrtombs.o             \
804         wcswidth.o              \
805         wcstombs.o              \
806         wcsxfrm.o               \
807         wctob.o                 \
808         wctomb.o                \
809         wctrans.o               \
810         wctype.o                \
811         wcwidth.o               \
812         wscol.o

814 AIOOBJS=                       \
815         aio.o                   \
816         aio_alloc.o             \
817         posix_aio.o

819 RTOBJS=                        \
820         clock_timer.o           \
821         mqueue.o                \
822         pos4obj.o               \
823         sched.o                 \
824         sem.o                   \
825         shm.o                   \
826         sigev_thread.o

828 SECFLAGSOBJS=                  \
829         secflags.o

831 TPOOLOBJS=                     \
832         thread_pool.o

834 THREADSOBJS=                   \
835         alloc.o                 \
836         assfail.o               \
837         c11_thr.o               \
838         cancel.o                \
839         door_calls.o            \
840         tmem.o                  \
841         pthr_attr.o             \
842         pthr_barrier.o          \
843         pthr_cond.o             \
844         pthr_mutex.o            \
845         pthr_rwlock.o           \
846         pthread.o               \
847         rwlock.o                \
848         scalls.o                \
849         sema.o                  \
850         sigaction.o             \
851         spawn.o                 \
852         synch.o                 \
```



```
853         tdb_agent.o             \
854         thr.o                   \
855         thread_interface.o      \
856         tls.o                   \
857         tsd.o

859 THREADSMACHOBJS=               \
860         machdep.o

862 THREADSASMOBJS=                \
863         asm_subr.o

865 UNICODEOBJS=                   \
866         u8_textprep.o           \
867         uconv.o

869 UNWINDMACHOBJS=                \
870         call_frame_inst.o       \
871         eh_frame.o              \
872         thrp_unwind.o           \
873         unwind.o

875 pics/unwind.o:= COPTFLAG64 =

877 UNWINDASMOBJS=                 \
878         unwind_frame.o

880 # Preserved solely to ease maintenance of 32-bit and 64-bit library builds
881 # This macro should ALWAYS be empty; native APIs are already 'large file'.
882 PORTSYS64=

884 PORTSYS=                       \
885         _autofssys.o            \
886         access.o                \
887         acctctl.o               \
888         bsd_signal.o            \
889         chmod.o                 \
890         chown.o                 \
891         corectl.o               \
892         epoll.o                 \
893         exacctsys.o             \
894         execl.o                 \
895         execle.o                \
896         execv.o                 \
897         eventfd.o               \
898         fcntl.o                 \
899         getpagesizes.o          \
900         getpeerucred.o          \
901         inst_sync.o             \
902         issetugid.o             \
903         label.o                 \
904         link.o                  \
905         lockf.o                 \
906         lwp.o                   \
907         lwp_cond.o              \
908         lwp_rwlock.o            \
909         lwp_sigmask.o           \
910         meminfosys.o            \
911         mkdir.o                 \
912         mknod.o                 \
913         msgsys.o                \
914         nfssys.o                \
915         open.o                  \
916         pgrpsys.o               \
917         posix_sigwait.o         \
918         ppriv.o                 \
```

```
 919         psetsys.o               \
 920         rctlsys.o               \
 921         readlink.o              \
 922         rename.o                \
 923         sbrk.o                  \
 924         semsys.o                \
 925         set_errno.o             \
 926         sharefs.o               \
 927         shmsys.o                \
 928         sidsys.o                \
 929         siginterrupt.o          \
 930         signal.o                \
 931         signalfd.o              \
 932         sigpending.o            \
 933         sigstack.o              \
 934         stat.o                  \
 935         symlink.o               \
 936         tasksys.o               \
 937         time.o                  \
 938         time_util.o             \
 939         timerfd.o               \
 940         ucontext.o              \
 941         unlink.o                \
 942         ustat.o                 \
 943         utimesys.o              \
 944         zone.o

 946 PORTREGEX=                      \
 947         glob.o                  \
 948         regcmp.o                \
 949         regcomp.o               \
 950         regerror.o              \
 951         regex.o                 \
 952         regexec.o               \
 953         regfree.o               \
 954         wordexp.o

 956 VALUES=                        \
 957         values-Xa.o

 959 MOSTOBJS=                      \
 960         $(STRETS)               \
 961         $(CRTOBJS)              \
 962         $(DYNOBJS)              \
 963         $(FPOBJS)               \
 964         $(I386FPOBJS)           \
 965         $(FPASMOBJS)            \
 966         $(ATOMICOBJS)           \
 967         $(CHACHAOBJS)           \
 968         $(XATTROBJS)            \
 969         $(COMOBJS)              \
 970         $(GENOBJS)              \
 971         $(PORTFP)               \
 972         $(PORTGEN)              \
 973         $(PORTGEN64)            \
 974         $(PORTI18N)             \
 975         $(PORTI18N_COND)        \
 976         $(PORTINET)             \
 977         $(PORTLOCALE)           \
 978         $(PORTPRINT)            \
 979         $(PORTPRINT_W)          \
 980         $(PORTREGEX)            \
 981         $(PORTSTDIO)            \
 982         $(PORTSTDIO64)          \
 983         $(PORTSTDIO_W)          \
 984         $(PORTSYS)              \
```



```
 985         $(PORTSYS64)            \
 986         $(AIOOBJS)              \
 987         $(RTOBJS)               \
 988         $(SECFLAGSOBJS)         \
 989         $(TPOOLOBJS)            \
 990         $(THREADSOBJS)          \
 991         $(THREADSMACHOBJS)      \
 992         $(THREADSASMOBJS)       \
 993         $(UNICODEOBJS)          \
 994         $(UNWINDMACHOBJS)       \
 995         $(UNWINDASMOBJS)        \
 996         $(COMSYSOBJS)           \
 997         $(SYSOBJS)              \
 998         $(COMSYSOBJS64)         \
 999         $(SYSOBJS64)            \
1000         $(VALUES)

1002 TRACEOBJS=                      \
1003         plockstat.o

1005 # NOTE: libc.so.1 must be linked with the minimal crti.o and crtn.o
1006 # modules whose source is provided in the $(SRC)/lib/crt directory.
1007 # This must be done because otherwise the Sun C compiler would insert
1008 # its own versions of these modules and those versions contain code
1009 # to call out to C++ initialization functions.  Such C++ initialization
1010 # functions can call back into libc before thread initialization is
1011 # complete and this leads to segmentation violations and other problems.
1012 # Since libc contains no C++ code, linking with the minimal crti.o and
1013 # crtn.o modules is safe and avoids the problems described above.
1014 OBJECTS= $(CRTI) $(MOSTOBJS) $(CRTN)
1015 CRTSRCS= ../../crt/amd64

1017 # include common library definitions
1018 include ../../Makefile.lib
1019 include ../../Makefile.lib.64

1021 CFLAGS64 += $(CTF_FLAGS)

1023 # This is necessary to avoid problems with calling _ex_unwind().
1024 # We probably don't want any inlining anyway.
1025 CFLAGS64 += -xinline=

1027 CERRWARN += -_gcc=-Wno-parentheses
1028 CERRWARN += -_gcc=-Wno-switch
1029 CERRWARN += -_gcc=-Wno-uninitialized
1030 CERRWARN += -_gcc=-Wno-unused-value
1031 CERRWARN += -_gcc=-Wno-unused-label
1032 CERRWARN += -_gcc=-Wno-unused-variable
1033 CERRWARN += -_gcc=-Wno-type-limits
1034 CERRWARN += -_gcc=-Wno-char-subscripts
1035 CERRWARN += -_gcc=-Wno-clobbered
1036 CERRWARN += -_gcc=-Wno-unused-function
1037 CERRWARN += -_gcc=-Wno-address

1039 # not linted
1040 SMATCH=off

1042 # Setting THREAD_DEBUG = -DTHREAD_DEBUG (make THREAD_DEBUG=-DTHREAD_DEBUG ...)
1043 # enables ASSERT() checking in the threads portion of the library.
1044 # This is automatically enabled for DEBUG builds, not for non-debug builds.
1045 THREAD_DEBUG =
1046 $(NOT_RELEASE_BUILD)THREAD_DEBUG = -DTHREAD_DEBUG

1048 # Make string literals read-only to save memory
1049 CFLAGS64 += $(XSTRCONST)
```

```
1051 ALTPICS= $(TRACEOBJS:%=pics/%)

1053 $(DYNLIB) := BUILD.SO = $(LD) -o $@ -G $(DYNFLAGS) $(PICS) $(ALTPICS) $(EXTPICS)

1055 MAPFILES =      $(LIBCDIR)/port/mapfile-vers

1057 CPPFLAGS=       -D_REENTRANT -D$(MACH64) -D__$(MACH64) $(THREAD_DEBUG) \
1058                 -I. -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc $(CPPFLAGS.master)
1059 ASFLAGS=        $(AS_PICFLAGS) -P -D__STDC__ -D_ASM $(CPPFLAGS) \
1060                 $(amd64_AS_XARCH)

1062 # As a favor to the dtrace syscall provider, libc still calls the
1063 # old syscall traps that have been obsoleted by the *at() interfaces.
1064 # Delete this to compile libc using only the new *at() system call traps
1065 CPPFLAGS += -D_RETAIN_OLD_SYSCALLS

1067 # proc64_id.o is built with defines in $(SRC)/uts/intel/sys/x86_archext.h
1068 pics/proc64_id.o        := CFLAGS64 += -I$(SRC)/uts/intel

1070 # Inform the run-time linker about libc specialized initialization
1071 RTLDINFO =      -z rtldinfo=tls_rtldinfo
1072 DYNFLAGS +=     $(RTLDINFO)

1074 # Force libc's internal references to be resolved immediately upon loading
1075 # in order to avoid critical region problems.  Since almost all libc symbols
1076 # are marked 'protected' in the mapfiles, this is a minimal set (15 to 20).
1077 DYNFLAGS +=     -znow

1079 BUILD.s=        $(AS) $(ASFLAGS) $< -o $@

1081 # Override this top level flag so the compiler builds in its native
1082 # C99 mode.  This has been enabled to support the complex arithmetic
1083 # added to libc.
1084 CSTD=   $(CSTD_GNU99)

1086 # libc method of building an archive
1087 # The "$(GREP) -v ' L '" part is necessary only until
1088 # lorder is fixed to ignore thread-local variables.
1089 BUILD.AR= $(RM) $@ ; \
1090         $(AR) q $@ `$(LORDER) $(MOSTOBJS:%=$(DIR)/%) | $(GREP) -v ' L ' | $(TSOR

1092 # extra files for the clean target
1093 CLEANFILES +=                   \
1094         $(LIBCDIR)/port/gen/errlst.c    \
1095         $(LIBCDIR)/port/gen/new_list.c  \
1096         assym.h                 \
1097         genassym                \
1098         crt/_rtld.s             \
1099         pics/crti.o             \
1100         pics/crtn.o             \
1101         $(ALTPICS)

1103 CLOBBERFILES += $(LIB_PIC)

1105 # list of C source for lint
1106 SRCS=                                                   \
1107         $(ATOMICOBJS:%.o=$(SRC)/common/atomic/%.c)      \
1108         $(XATTROBJS:%.o=$(SRC)/common/xattr/%.c)        \
1109         $(COMOBJS:%.o=$(SRC)/common/util/%.c)           \
1110         $(PORTFP:%.o=$(LIBCDIR)/port/fp/%.c)            \
1111         $(PORTGEN:%.o=$(LIBCDIR)/port/gen/%.c)          \
1112         $(PORTI18N:%.o=$(LIBCDIR)/port/i18n/%.c)        \
1113         $(PORTINET:%.o=$(LIBCDIR)/port/inet/%.c)        \
1114         $(PORTLOCALE:%.o=$(LIBCDIR)/port/locale/%.c)    \
1115         $(PORTPRINT:%.o=$(LIBCDIR)/port/print/%.c)      \
1116         $(PORTREGEX:%.o=$(LIBCDIR)/port/regex/%.c)      \
```



```
1117         $(PORTSTDIO:%.o=$(LIBCDIR)/port/stdio/%.c)      \
1118         $(PORTSYS:%.o=$(LIBCDIR)/port/sys/%.c)          \
1119         $(AIOOBJS:%.o=$(LIBCDIR)/port/aio/%.c)          \
1120         $(RTOBJS:%.o=$(LIBCDIR)/port/rt/%.c)            \
1121         $(SECFLAGSOBJS:%.o=$(SRC)/common/secflags/%.c)  \
1122         $(TPOOLOBJS:%.o=$(LIBCDIR)/port/tpool/%.c)      \
1123         $(THREADSOBJS:%.o=$(LIBCDIR)/port/threads/%.c)  \
1124         $(THREADSMACHOBJS:%.o=threads/%.c)              \
1125         $(UNICODEOBJS:%.o=$(SRC)/common/unicode/%.c)    \
1126         $(UNWINDMACHOBJS:%.o=unwind/%.c)                \
1127         $(FPOBJS:%.o=fp/%.c)                            \
1128         $(I386FPOBJS:%.o=$(LIBCDIR)/i386/fp/%.c)        \
1129         $(LIBCBASE)/gen/ecvt.c                          \
1130         $(LIBCBASE)/gen/makectxt.c                      \
1131         $(LIBCBASE)/gen/siginfolst.c                    \
1132         $(LIBCBASE)/gen/siglongjmp.c                    \
1133         $(LIBCBASE)/gen/sync_instruction_memory.c       \
1134         $(LIBCBASE)/sys/uadmin.c

1136 # conditional assignments
1137 # $(DYNLIB) $(LIB_PIC) := DYNOBJS = _rtbootld.o
1138 $(DYNLIB) := CRTI = crti.o
1139 $(DYNLIB) := CRTN = crtn.o

1141 # Files which need the threads .il inline template
1142 TIL=                            \
1143         aio.o                   \
1144         alloc.o                 \
1145         assfail.o               \
1146         atexit.o                \
1147         atfork.o                \
1148         cancel.o                \
1149         door_calls.o            \
1150         err.o                   \
1151         errno.o                 \
1152         lwp.o                   \
1153         ma.o                    \
1154         machdep.o               \
1155         posix_aio.o             \
1156         pthr_attr.o             \
1157         pthr_barrier.o          \
1158         pthr_cond.o             \
1159         pthr_mutex.o            \
1160         pthr_rwlock.o           \
1161         pthread.o               \
1162         rand.o                  \
1163         rwlock.o                \
1164         scalls.o                \
1165         sched.o                 \
1166         sema.o                  \
1167         sigaction.o             \
1168         sigev_thread.o          \
1169         spawn.o                 \
1170         stack.o                 \
1171         synch.o                 \
1172         tdb_agent.o             \
1173         thr.o                   \
1174         thread_interface.o      \
1175         thread_pool.o           \
1176         thrp_unwind.o           \
1177         tls.o                   \
1178         tmem.o                  \
1179         tsd.o

1181 $(TIL:%=pics/%) := CFLAGS64 += $(LIBCBASE)/threads/amd64.il
```

```
1183 # pics/mul64.o := CFLAGS64 += crt/mul64.il

1185 # large-file-aware components that should be built large

1187 #$(COMSYSOBJS64:%=pics/%) := \
1188 #       CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1190 #$(SYSOBJS64:%=pics/%) := \
1191 #       CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1193 #$(PORTGEN64:%=pics/%) := \
1194 #       CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1196 #$(PORTSTDIO64:%=pics/%) := \
1197 #       CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1199 #$(PORTSYS64:%=pics/%) := \
1200 #       CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1202 $(PORTSTDIO_W:%=pics/%) := \
1203         CPPFLAGS += -D_WIDE

1205 $(PORTPRINT_W:%=pics/%) := \
1206         CPPFLAGS += -D_WIDE

1208 $(PORTPRINT_C89:%=pics/%) := \
1209         CPPFLAGS += -D_C89_INTMAX32

1211 $(PORTSTDIO_C89:%=pics/%) := \
1212         CPPFLAGS += -D_C89_INTMAX32

1214 $(PORTI18N_COND:%=pics/%) := \
1215         CPPFLAGS += -D_WCS_LONGLONG

1217 pics/arc4random.o :=    CPPFLAGS += -I$(SRC)/common/crypto/chacha

1219 pics/__clock_gettime.o := CPPFLAGS += $(COMMPAGE_CPPFLAGS)

1221 .KEEP_STATE:

1223 all: $(LIBS) $(LIB_PIC)

1225 lint    :=      CPPFLAGS += -I$(LIBCDIR)/$(MACH)/fp
1226 lint    :=      CPPFLAGS += -D_MSE_INT_H -D_LCONV_C99
1227 lint    :=      LINTFLAGS64 += -mn -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED

1229 lint:
1230         @echo $(LINT.c) ... $(LDLIBS)
1231         @$(LINT.c) $(SRCS) $(LDLIBS)

1233 $(LINTLIB):= SRCS=$(LIBCDIR)/port/llib-lc
1234 $(LINTLIB):= CPPFLAGS += -D_MSE_INT_H
1235 $(LINTLIB):= LINTFLAGS64=-nvx -m64

1237 # object files that depend on inline template
1238 $(TIL:%=pics/%): $(LIBCBASE)/threads/amd64.il
1239 # pics/mul64.o: crt/mul64.il

1241 # include common libc targets
1242 include ../Makefile.targ

1244 # We need to strip out all CTF data from the static library
1245 $(LIB_PIC) := DIR = pics
1246 $(LIB_PIC): pics $$(PICS)
1247         $(BUILD.AR)
1248         $(MCS) -d -n .SUNW_ctf $@ > /dev/null 2>&1
```



```
1249         $(AR) -ts $@ > /dev/null
1250         $(POST_PROCESS_A)

1252 ASSYMDEP_OBJS=                  \
1253         _lwp_mutex_unlock.o     \
1254         _stack_grow.o           \
1255         asm_subr.o              \
1256         getcontext.o            \
1257         setjmp.o                \
1258         tls_get_addr.o          \
1259         vforkx.o

1261 $(ASSYMDEP_OBJS:%=pics/%): assym.h

1263 # assym.h build rules

1265 GENASSYM_C = genassym.c

1267 genassym: $(GENASSYM_C)
1268         $(NATIVECC) $(NATIVE_CFLAGS) -Iinc -I$(LIBCDIR)/inc $(CPPFLAGS.native) \
1269                 -o $@ $(GENASSYM_C)

1271 OFFSETS = $(LIBCDIR)/$(MACH)/offsets.in

1273 assym.h: $(OFFSETS) genassym
1274         $(OFFSETS_CREATE) <$(OFFSETS) >$@
1275         ./genassym >>$@

1277 # derived C source and related explicit dependencies
1278 $(LIBCDIR)/port/gen/errlst.c + \
1279 $(LIBCDIR)/port/gen/new_list.c: $(LIBCDIR)/port/gen/errlist $(LIBCDIR)/port/gen/
1280         cd $(LIBCDIR)/port/gen; pwd; $(AWK) -f errlist.awk < errlist

1282 pics/errlst.o: $(LIBCDIR)/port/gen/errlst.c

1284 pics/new_list.o: $(LIBCDIR)/port/gen/new_list.c
```

```
**********************************************************
    1372 Fri Dec 21 14:59:56 2018
new/usr/src/lib/libc/i386/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2007 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 # Copyright (c) 2018, Joyent, Inc.
  25 # ident "%Z%%M% %I%     %E% SMI"
  26 #
  27 # lib/libc/i386/Makefile
  28 #

  27 LIBCBASE=../i386

  29 EXTN_CPPFLAGS =
  30 EXTN_XFLAGS =

  32 LIBRARY =       libc.a

  34 # Special postprocessing for 32-bit fpcw.o: strip the SSE HWCAP attribute
  35 # from the object file; the internal functions with SSE instructions are
  36 # called conditionally on systems which have SSE instruction support
  37 pics/fpcw.o := POST_PROCESS_O += ; \
  38         if $(ELFDUMP) -H $@ | grep " SSE "; then \
  39                 $(ELFEDIT) -e 'cap:hw1 -and -cmp SSE' $@; \
  40         fi

  42 include Makefile.com
```

```
**********************************************************
   25161 Fri Dec 21 14:59:57 2018
new/usr/src/lib/libc/i386/Makefile.com
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #

  22 #
  23 # Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
  24 # Copyright 2016 Joyent, Inc.
  25 # Copyright (c) 2013, OmniTI Computer Consulting, Inc. All rights reserved.
  26 # Copyright 2013 Garrett D'Amore <garrett@damore.org>
  27 # Copyright 2018 Nexenta Systems, Inc.
  28 #

  30 LIBCDIR=        $(SRC)/lib/libc
  31 LIB_PIC=        libc_pic.a
  32 VERS=           .1
  33 CPP=            /usr/lib/cpp
  34 TARGET_ARCH=    i386

  36 # include comm page definitions
  37 include $(SRC)/lib/commpage/Makefile.shared.com
  38 include $(SRC)/lib/commpage/Makefile.shared.targ

  40 VALUES=         values-Xa.o

  42 # objects are grouped by source directory

  44 # local objects
  45 STRETS=

  47 CRTOBJS=                       \
  48         cerror.o               \
  49         cerror64.o

  51 DYNOBJS=                       \
  52         _rtbootld.o

  54 FPOBJS=                        \
  55         _D_cplx_div.o          \
  56         _D_cplx_div_ix.o       \
  57         _D_cplx_div_rx.o       \
  58         _D_cplx_lr_div.o       \
  59         _D_cplx_lr_div_ix.o    \
  60         _D_cplx_lr_div_rx.o    \
```



```
  61         _D_cplx_mul.o          \
  62         _F_cplx_div.o          \
  63         _F_cplx_div_ix.o       \
  64         _F_cplx_div_rx.o       \
  65         _F_cplx_lr_div.o       \
  66         _F_cplx_lr_div_ix.o    \
  67         _F_cplx_lr_div_rx.o    \
  68         _F_cplx_mul.o          \
  69         _X_cplx_div.o          \
  70         _X_cplx_div_ix.o       \
  71         _X_cplx_div_rx.o       \
  72         _X_cplx_lr_div.o       \
  73         _X_cplx_lr_div_ix.o    \
  74         _X_cplx_lr_div_rx.o    \
  75         _X_cplx_mul.o          \
  76         fpgetmask.o            \
  77         fpgetround.o           \
  78         fpgetsticky.o          \
  79         fpsetmask.o            \
  80         fpsetround.o           \
  81         fpsetsticky.o          \
  82         fpstart.o              \
  83         ieee.o

  85 FPASMOBJS=                     \
  86         __xgetRD.o             \
  87         _base_il.o             \
  88         _xtoll.o               \
  89         _xtoull.o              \
  90         fpcw.o

  92 ATOMICOBJS=                   \
  93         atomic.o

  95 CHACHAOBJS=                   \
  96         chacha.o

  98 XATTROBJS=                    \
  99         xattr_common.o

 101 COMOBJS=                      \
 102         bcmp.o                 \
 103         bcopy.o                \
 104         bsearch.o              \
 105         bzero.o                \
 106         qsort.o                \
 107         strtol.o               \
 108         strtoul.o              \
 109         strtoll.o              \
 110         strtoull.o

 112 DTRACEOBJS=                    \
 113         dtrace_data.o

 115 SECFLAGSOBJS=                 \
 116         secflags.o

 118 GENOBJS=                      \
 119         $(COMMPAGE_OBJS)       \
 120         _div64.o               \
 121         _divdi3.o              \
 122         _getsp.o               \
 123         _mul64.o               \
 124         abs.o                  \
 125         alloca.o               \
 126         arc4random.o           \
```

```
127         arc4random_uniform.o    \
128         byteorder.o             \
129         byteorder64.o           \
130         cuexit.o                \
131         ecvt.o                  \
132         endian.o                \
133         errlst.o                \
134         i386_data.o             \
135         ladd.o                  \
136         ldivide.o               \
137         lmul.o                  \
138         lock.o                  \
139         lshiftl.o               \
140         lsign.o                 \
141         lsub.o                  \
142         makectxt.o              \
143         memccpy.o               \
144         memchr.o                \
145         memcmp.o                \
146         memcpy.o                \
147         memset.o                \
148         new_list.o              \
149         setjmp.o                \
150         siginfolst.o            \
151         siglongjmp.o            \
152         strcat.o                \
153         strchr.o                \
154         strcmp.o                \
155         strcpy.o                \
156         strlen.o                \
157         strncat.o               \
158         strncmp.o               \
159         strncpy.o               \
160         strnlen.o               \
161         strrchr.o               \
162         sync_instruction_memory.o

164 # sysobjs that contain large-file interfaces
165 COMSYSOBJS64=                   \
166         fstatvfs64.o            \
167         getdents64.o            \
168         getrlimit64.o           \
169         lseek64.o               \
170         mmap64.o                \
171         pread64.o               \
172         preadv64.o              \
173         pwrite64.o              \
174         pwritev64.o             \
175         setrlimit64.o           \
176         statvfs64.o

178 SYSOBJS64=

180 COMSYSOBJS=                     \
181         __clock_timer.o         \
182         __getloadavg.o          \
183         __rusagesys.o           \
184         __signotify.o           \
185         __sigrt.o               \
186         __time.o                \
187         _lgrp_home_fast.o       \
188         _lgrpsys.o              \
189         _nfssys.o               \
190         _portfs.o               \
191         _pset.o                 \
192         _rpcsys.o               \
```



```
193         _sigaction.o            \
194         _so_accept.o            \
195         _so_bind.o              \
196         _so_connect.o           \
197         _so_getpeername.o       \
198         _so_getsockname.o       \
199         _so_getsockopt.o        \
200         _so_listen.o            \
201         _so_recv.o              \
202         _so_recvfrom.o          \
203         _so_recvmsg.o           \
204         _so_send.o              \
205         _so_sendmsg.o           \
206         _so_sendto.o            \
207         _so_setsockopt.o        \
208         _so_shutdown.o          \
209         _so_socket.o            \
210         _so_socketpair.o        \
211         _sockconfig.o           \
212         acct.o                  \
213         acl.o                   \
214         adjtime.o               \
215         alarm.o                 \
216         brk.o                   \
217         chdir.o                 \
218         chroot.o                \
219         cladm.o                 \
220         close.o                 \
221         execve.o                \
222         exit.o                  \
223         facl.o                  \
224         fchdir.o                \
225         fchroot.o               \
226         fdsync.o                \
227         fpathconf.o             \
228         fstatfs.o               \
229         fstatvfs.o              \
230         getcpuid.o              \
231         getdents.o              \
232         getegid.o               \
233         geteuid.o               \
234         getgid.o                \
235         getgroups.o             \
236         gethrtime.o             \
237         getitimer.o             \
238         getmsg.o                \
239         getpid.o                \
240         getpmsg.o               \
241         getppid.o               \
242         getrandom.o             \
243         getrlimit.o             \
244         getuid.o                \
245         gtty.o                  \
246         install_utrap.o         \
247         ioctl.o                 \
248         kaio.o                  \
249         kill.o                  \
250         llseek.o                \
251         lseek.o                 \
252         mmapobjsys.o            \
253         memcntl.o               \
254         mincore.o               \
255         mmap.o                  \
256         modctl.o                \
257         mount.o                 \
258         mprotect.o              \
```

```
259         munmap.o                \
260         nice.o                  \
261         ntp_adjtime.o           \
262         ntp_gettime.o           \
263         p_online.o              \
264         pathconf.o              \
265         pause.o                 \
266         pcsample.o              \
267         pipe2.o                 \
268         pollsys.o               \
269         pread.o                 \
270         preadv.o                \
271         priocntlset.o           \
272         processor_bind.o        \
273         processor_info.o        \
274         profil.o                \
275         psecflagsset.o          \
276         putmsg.o                \
277         putpmsg.o               \
278         pwrite.o                \
279         pwritev.o               \
280         read.o                  \
281         readv.o                 \
282         resolvepath.o           \
283         seteguid.o              \
284         setgid.o                \
285         setgroups.o             \
286         setitimer.o             \
287         setreid.o               \
288         setrlimit.o             \
289         setuid.o                \
290         sigaltstk.o             \
291         sigprocmsk.o            \
292         sigsendset.o            \
293         sigsuspend.o            \
294         statfs.o                \
295         statvfs.o               \
296         stty.o                  \
297         sync.o                  \
298         sysconfig.o             \
299         sysfs.o                 \
300         sysinfo.o               \
301         syslwp.o                \
302         times.o                 \
303         ulimit.o                \
304         umask.o                 \
305         umount2.o               \
306         utssys.o                \
307         uucopy.o                \
308         vhangup.o               \
309         waitid.o                \
310         write.o                 \
311         writev.o                \
312         yield.o

314 SYSOBJS=                       \
315         __clock_gettime.o       \
316         __clock_gettime_sys.o   \
317         __getcontext.o          \
318         __uadmin.o              \
319         _lwp_mutex_unlock.o     \
320         _stack_grow.o           \
321         door.o                  \
322         forkx.o                 \
323         forkallx.o              \
324         getcontext.o            \
```



```
325         gettimeofday.o          \
326         lwp_private.o           \
327         nuname.o                \
328         ptrace.o                \
329         syscall.o               \
330         sysi86.o                \
331         tls_get_addr.o          \
332         uadmin.o                \
333         umount.o                \
334         uname.o                 \
335         vforkx.o                \
336         xstat.o

338 # objects under $(LIBCDIR)/port which contain transitional large file interfaces
339 PORTGEN64=                      \
340         _xftw64.o               \
341         attropen64.o            \
342         fts64.o                 \
343         ftw64.o                 \
344         mkstemp64.o             \
345         nftw64.o                \
346         tell64.o                \
347         truncate64.o

349 # objects from source under $(LIBCDIR)/port
350 PORTFP=                         \
351         __flt_decim.o           \
352         __flt_rounds.o          \
353         __tbl_10_b.o            \
354         __tbl_10_h.o            \
355         __tbl_10_s.o            \
356         __tbl_2_b.o             \
357         __tbl_2_h.o             \
358         __tbl_2_s.o             \
359         __tbl_fdq.o             \
360         __tbl_tens.o            \
361         __x_power.o             \
362         _base_sup.o             \
363         aconvert.o              \
364         decimal_bin.o           \
365         double_decim.o          \
366         econvert.o              \
367         fconvert.o              \
368         file_decim.o            \
369         finite.o                \
370         fp_data.o               \
371         func_decim.o            \
372         gconvert.o              \
373         hex_bin.o               \
374         ieee_globals.o          \
375         pack_float.o            \
376         sigfpe.o                \
377         string_decim.o

379 PORTGEN=                       \
380         _env_data.o             \
381         _xftw.o                 \
382         a64l.o                  \
383         abort.o                 \
384         addsev.o                \
385         ascii_strcasecmp.o      \
386         ascii_strncasecmp.o     \
387         assert.o                \
388         atexit.o                \
389         atfork.o                \
390         atof.o                  \
```

```
391         atoi.o                  \
392         atol.o                  \
393         atoll.o                 \
394         attrat.o                \
395         attropen.o              \
396         basename.o              \
397         calloc.o                \
398         catgets.o               \
399         catopen.o               \
400         cfgetispeed.o           \
401         cfgetospeed.o           \
402         cfree.o                 \
403         cfsetispeed.o           \
404         cfsetospeed.o           \
405         cftime.o                \
406         clock.o                 \
407         closedir.o              \
408         closefrom.o             \
409         confstr.o               \
410         crypt.o                 \
411         csetlen.o               \
412         ctime.o                 \
413         ctime_r.o               \
414         daemon.o                \
415         deflt.o                 \
416         directio.o              \
417         dirname.o               \
418         div.o                   \
419         drand48.o               \
420         dup.o                   \
421         env_data.o              \
422         err.o                   \
423         errno.o                 \
424         euclen.o                \
425         event_port.o            \
426         execvp.o                \
427         explicit_bzero.o        \
428         fattach.o               \
429         fdetach.o               \
430         fdopendir.o             \
431         ffs.o                   \
432         flock.o                 \
433         fls.o                   \
434         fmtmsg.o                \
435         freezero.o              \
436         ftime.o                 \
437         ftok.o                  \
438         fts.o                   \
439         ftw.o                   \
440         gcvt.o                  \
441         get_nprocs.o            \
442         getauxv.o               \
443         getcwd.o                \
444         getdate_err.o           \
445         getdtblsize.o           \
446         getentropy.o            \
447         getenv.o                \
448         getexecname.o           \
449         getgrnam.o              \
450         getgrnam_r.o            \
451         gethostid.o             \
452         gethostname.o           \
453         gethz.o                 \
454         getisax.o               \
455         getloadavg.o            \
456         getlogin.o              \
```



```
457         getmntent.o             \
458         getnetgrent.o           \
459         getopt.o                \
460         getopt_long.o           \
461         getpagesize.o           \
462         getpw.o                 \
463         getpwnam.o              \
464         getpwnam_r.o            \
465         getrusage.o             \
466         getspent.o              \
467         getspent_r.o            \
468         getsubopt.o             \
469         gettxt.o                \
470         getusershell.o          \
471         getut.o                 \
472         getutx.o                \
473         getvfsent.o             \
474         getwd.o                 \
475         getwidth.o              \
476         getxby_door.o           \
477         gtxt.o                  \
478         hsearch.o               \
479         iconv.o                 \
480         imaxabs.o               \
481         imaxdiv.o               \
482         index.o                 \
483         initgroups.o            \
484         insque.o                \
485         isaexec.o               \
486         isastream.o             \
487         isatty.o                \
488         killpg.o                \
489         klpdlib.o               \
490         l64a.o                  \
491         lckpwdf.o               \
492         lconstants.o            \
493         lexp10.o                \
494         lfind.o                 \
495         lfmt.o                  \
496         lfmt_log.o              \
497         llabs.o                 \
498         lldiv.o                 \
499         llog10.o                \
500         lltostr.o               \
501         localtime.o             \
502         lsearch.o               \
503         madvise.o               \
504         malloc.o                \
505         memalign.o              \
506         memmem.o                \
507         memset_s.o              \
508         mkdev.o                 \
509         mkdtemp.o               \
510         mkfifo.o                \
511         mkstemp.o               \
512         mktemp.o                \
513         mlock.o                 \
514         mlockall.o              \
515         mon.o                   \
516         msync.o                 \
517         munlock.o               \
518         munlockall.o            \
519         ndbm.o                  \
520         nftw.o                  \
521         nlspath_checks.o        \
522         nsparse.o               \
```

```
523         nss_common.o            \
524         nss_dbdefs.o            \
525         nss_deffinder.o         \
526         opendir.o               \
527         opt_data.o              \
528         perror.o                \
529         pfmt.o                  \
530         pfmt_data.o             \
531         pfmt_print.o            \
532         pipe.o                  \
533         plock.o                 \
534         poll.o                  \
535         posix_fadvise.o         \
536         posix_fallocate.o       \
537         posix_madvise.o         \
538         posix_memalign.o        \
539         priocntl.o              \
540         priv_str_xlate.o        \
541         privlib.o               \
542         psecflags.o             \
543         psiginfo.o              \
544         psignal.o               \
545         pt.o                    \
546         putpwent.o              \
547         putspent.o              \
548         raise.o                 \
549         rand.o                  \
550         random.o                \
551         rctlops.o               \
552         readdir.o               \
553         readdir_r.o             \
554         reallocarray.o          \
555         recallocarray.o         \
556         realpath.o              \
557         reboot.o                \
558         regexpr.o               \
559         remove.o                \
560         rewinddir.o             \
561         rindex.o                \
562         scandir.o               \
563         seekdir.o               \
564         select.o                \
565         set_constraint_handler_s.o \
566         setlabel.o              \
567         setpriority.o           \
568         settimeofday.o          \
569         sh_locks.o              \
570         sigflag.o               \
571         siglist.o               \
572         sigsend.o               \
573         sigsetops.o             \
574         ssignal.o               \
575         stack.o                 \
576         stpcpy.o                \
577         stpncpy.o               \
578         str2sig.o               \
579         strcase_charmap.o       \
580         strchrnul.o             \
581         strcspn.o               \
582         strdup.o                \
583         strerror.o              \
584         strlcat.o               \
585         strlcpy.o               \
586         strndup.o               \
587         strpbrk.o               \
588         strsep.o                \
```



```
589         strsignal.o             \
590         strspn.o                \
591         strstr.o                \
592         strtod.o                \
593         strtoimax.o             \
594         strtok.o                \
595         strtok_r.o              \
596         strtonum.o              \
597         strtoumax.o             \
598         swab.o                  \
599         swapctl.o               \
600         sysconf.o               \
601         syslog.o                \
602         tcdrain.o               \
603         tcflow.o                \
604         tcflush.o               \
605         tcgetattr.o             \
606         tcgetpgrp.o             \
607         tcgetsid.o              \
608         tcsendbreak.o           \
609         tcsetattr.o             \
610         tcsetpgrp.o             \
611         tell.o                  \
612         telldir.o               \
613         tfind.o                 \
614         time_data.o             \
615         time_gdata.o            \
616         timespec_get.o          \
617         tls_data.o              \
618         truncate.o              \
619         tsdalloc.o              \
620         tsearch.o               \
621         ttyname.o               \
622         ttyslot.o               \
623         ualarm.o                \
624         ucred.o                 \
625         valloc.o                \
626         vlfmt.o                 \
627         vpfmt.o                 \
628         waitpid.o               \
629         walkstack.o             \
630         wdata.o                 \
631         xgetwidth.o             \
632         xpg4.o                  \
633         xpg6.o

635 PORTINET=                       \
636         inet_lnaof.o            \
637         inet_makeaddr.o         \
638         inet_network.o          \
639         inet_ntoa.o             \
640         inet_ntop.o             \
641         inet_pton.o

643 PORTPRINT_W=                    \
644         doprnt_w.o

646 PORTPRINT=                     \
647         asprintf.o              \
648         doprnt.o                \
649         fprintf.o               \
650         printf.o                \
651         snprintf.o              \
652         sprintf.o               \
653         vfprintf.o              \
654         vprintf.o               \
```

```
655         vsnprintf.o             \
656         vsprintf.o              \
657         vwprintf.o              \
658         wprintf.o

660 # c89 variants to support 32-bit size of c89 u/intmax_t (32-bit libc only)
661 PORTPRINT_C89=                  \
662         vfprintf_c89.o          \
663         vprintf_c89.o           \
664         vsnprintf_c89.o         \
665         vsprintf_c89.o          \
666         vwprintf_c89.o

668 PORTSTDIO_C89=                  \
669         vscanf_c89.o            \
670         vwscanf_c89.o           \

672 # portable stdio objects that contain large file interfaces.
673 # Note: fopen64 is a special case, as we build it small.
674 PORTSTDIO64=                    \
675         fopen64.o               \
676         fpos64.o

678 PORTSTDIO_W=                    \
679         doscan_w.o

681 PORTSTDIO=                      \
682         __extensions.o          \
683         _endopen.o              \
684         _filbuf.o               \
685         _findbuf.o              \
686         _flsbuf.o               \
687         _wrtchk.o               \
688         clearerr.o              \
689         ctermid.o               \
690         ctermid_r.o             \
691         cuserid.o               \
692         data.o                  \
693         doscan.o                \
694         fdopen.o                \
695         feof.o                  \
696         ferror.o                \
697         fgetc.o                 \
698         fgets.o                 \
699         fileno.o                \
700         flockf.o                \
701         flush.o                 \
702         fopen.o                 \
703         fpos.o                  \
704         fputc.o                 \
705         fputs.o                 \
706         fread.o                 \
707         fseek.o                 \
708         fseeko.o                \
709         ftell.o                 \
710         ftello.o                \
711         fwrite.o                \
712         getc.o                  \
713         getchar.o               \
714         getline.o               \
715         getpass.o               \
716         gets.o                  \
717         getw.o                  \
718         mse.o                   \
719         popen.o                 \
720         putc.o                  \
```



```
721         putchar.o               \
722         puts.o                  \
723         putw.o                  \
724         rewind.o                \
725         scanf.o                 \
726         setbuf.o                \
727         setbuffer.o             \
728         setvbuf.o               \
729         system.o                \
730         tempnam.o               \
731         tmpfile.o               \
732         tmpnam_r.o              \
733         ungetc.o                \
734         vscanf.o                \
735         vwscanf.o               \
736         wscanf.o

738 PORTI18N=                       \
739         getwchar.o              \
740         putwchar.o              \
741         putws.o                 \
742         strtows.o               \
743         wcsnlen.o               \
744         wcsstr.o                \
745         wcstoimax.o             \
746         wcstol.o                \
747         wcstoul.o               \
748         wcswcs.o                \
749         wmemchr.o               \
750         wmemcmp.o               \
751         wmemcpy.o               \
752         wmemmove.o              \
753         wmemset.o               \
754         wscat.o                 \
755         wschr.o                 \
756         wscmp.o                 \
757         wscpy.o                 \
758         wscspn.o                \
759         wsdup.o                 \
760         wslen.o                 \
761         wsncat.o                \
762         wsncmp.o                \
763         wsncpy.o                \
764         wspbrk.o                \
765         wsprintf.o              \
766         wsrchr.o                \
767         wsscanf.o               \
768         wsspn.o                 \
769         wstod.o                 \
770         wstok.o                 \
771         wstol.o                 \
772         wstoll.o                \
773         wsxfrm.o                \
774         gettext.o               \
775         gettext_gnu.o           \
776         gettext_real.o          \
777         gettext_util.o          \
778         plural_parser.o         \
779         wdresolve.o             \
780         _ctype.o                \
781         isascii.o               \
782         toascii.o

784 PORTI18N_COND=                  \
785         wcstol_longlong.o       \
786         wcstoul_longlong.o
```

```
PORTLOCALE=                     \
        big5.o                  \
        btowc.o                 \
        collate.o               \
        collcmp.o               \
        euc.o                   \
        fnmatch.o               \
        fgetwc.o                \
        fgetws.o                \
        fix_grouping.o          \
        fputwc.o                \
        fputws.o                \
        fwide.o                 \
        gb18030.o               \
        gb2312.o                \
        gbk.o                   \
        getdate.o               \
        isdigit.o               \
        iswctype.o              \
        ldpart.o                \
        lmessages.o             \
        lnumeric.o              \
        lmonetary.o             \
        localeconv.o            \
        localeimpl.o            \
        mbftowc.o               \
        mblen.o                 \
        mbrlen.o                \
        mbrtowc.o               \
        mbsinit.o               \
        mbsnrtowcs.o            \
        mbsrtowcs.o             \
        mbstowcs.o              \
        mbtowc.o                \
        mskanji.o               \
        nextwctype.o            \
        nl_langinfo.o           \
        none.o                  \
        rune.o                  \
        runetype.o              \
        setlocale.o             \
        setrunelocale.o         \
        strcasecmp.o            \
        strcasestr.o            \
        strcoll.o               \
        strfmon.o               \
        strftime.o              \
        strncasecmp.o           \
        strptime.o              \
        strxfrm.o               \
        table.o                 \
        timelocal.o             \
        tolower.o               \
        towlower.o              \
        ungetwc.o               \
        utf8.o                  \
        wcrtomb.o               \
        wcscasecmp.o            \
        wcscoll.o               \
        wcsftime.o              \
        wcsnrtombs.o            \
        wcsrtombs.o             \
        wcswidth.o              \
        wcstombs.o              \
        wcsxfrm.o               \
```



```
        wctob.o                 \
        wctomb.o                \
        wctrans.o               \
        wctype.o                \
        wcwidth.o               \
        wscol.o

AIOOBJS=                        \
        aio.o                   \
        aio_alloc.o             \
        posix_aio.o

RTOBJS=                         \
        clock_timer.o           \
        mqueue.o                \
        pos4obj.o               \
        sched.o                 \
        sem.o                   \
        shm.o                   \
        sigev_thread.o

TPOOLOBJS=                      \
        thread_pool.o

THREADSOBJS=                    \
        alloc.o                 \
        assfail.o               \
        cancel.o                \
        c11_thr.o               \
        door_calls.o            \
        tmem.o                  \
        pthr_attr.o             \
        pthr_barrier.o          \
        pthr_cond.o             \
        pthr_mutex.o            \
        pthr_rwlock.o           \
        pthread.o               \
        rwlock.o                \
        scalls.o                \
        sema.o                  \
        sigaction.o             \
        spawn.o                 \
        synch.o                 \
        tdb_agent.o             \
        thr.o                   \
        thread_interface.o      \
        tls.o                   \
        tsd.o

THREADSMACHOBJS=                \
        machdep.o

THREADSASMOBJS=                 \
        asm_subr.o

UNICODEOBJS=                    \
        u8_textprep.o           \
        uconv.o

UNWINDMACHOBJS=                 \
        unwind.o

UNWINDASMOBJS=                  \
        unwind_frame.o

# objects that implement the transitional large file API
```

```
PORTSYS64=                      \
        lockf64.o               \
        stat64.o

PORTSYS=                        \
        _autofssys.o            \
        access.o                \
        acctctl.o               \
        bsd_signal.o            \
        chmod.o                 \
        chown.o                 \
        corectl.o               \
        epoll.o                 \
        eventfd.o               \
        exacctsys.o             \
        execl.o                 \
        execle.o                \
        execv.o                 \
        fcntl.o                 \
        getpagesizes.o          \
        getpeerucred.o          \
        inst_sync.o             \
        issetugid.o             \
        label.o                 \
        link.o                  \
        lockf.o                 \
        lwp.o                   \
        lwp_cond.o              \
        lwp_rwlock.o            \
        lwp_sigmask.o           \
        meminfosys.o            \
        mkdir.o                 \
        mknod.o                 \
        msgsys.o                \
        nfssys.o                \
        open.o                  \
        pgrpsys.o               \
        posix_sigwait.o         \
        ppriv.o                 \
        psetsys.o               \
        rctlsys.o               \
        readlink.o              \
        rename.o                \
        sbrk.o                  \
        semsys.o                \
        set_errno.o             \
        sharefs.o               \
        shmsys.o                \
        sidsys.o                \
        siginterrupt.o          \
        signal.o                \
        signalfd.o              \
        sigpending.o            \
        sigstack.o              \
        stat.o                  \
        symlink.o               \
        tasksys.o               \
        time.o                  \
        time_util.o             \
        timerfd.o               \
        ucontext.o              \
        unlink.o                \
        ustat.o                 \
        utimesys.o              \
        zone.o
```



```
PORTREGEX=                      \
        glob.o                  \
        regcmp.o                \
        regcomp.o               \
        regerror.o              \
        regex.o                 \
        regexec.o               \
        regfree.o               \
        wordexp.o

PORTREGEX64=                    \
        glob64.o

MOSTOBJS=                       \
        $(STRETS)               \
        $(CRTOBJS)              \
        $(DYNOBJS)              \
        $(FPOBJS)               \
        $(FPASMOBJS)            \
        $(ATOMICOBJS)           \
        $(CHACHAOBJS)           \
        $(XATTROBJS)            \
        $(COMOBJS)              \
        $(DTRACEOBJS)           \
        $(GENOBJS)              \
        $(PORTFP)               \
        $(PORTGEN)              \
        $(PORTGEN64)            \
        $(PORTI18N)             \
        $(PORTI18N_COND)        \
        $(PORTINET)             \
        $(PORTLOCALE)           \
        $(PORTPRINT)            \
        $(PORTPRINT_C89)        \
        $(PORTPRINT_W)          \
        $(PORTREGEX)            \
        $(PORTREGEX64)          \
        $(PORTSTDIO)            \
        $(PORTSTDIO64)          \
        $(PORTSTDIO_C89)        \
        $(PORTSTDIO_W)          \
        $(PORTSYS)              \
        $(PORTSYS64)            \
        $(AIOOBJS)              \
        $(RTOBJS)               \
        $(SECFLAGSOBJS)         \
        $(TPOOLOBJS)            \
        $(THREADSOBJS)          \
        $(THREADSMACHOBJS)      \
        $(THREADSASMOBJS)       \
        $(UNICODEOBJS)          \
        $(UNWINDMACHOBJS)       \
        $(UNWINDASMOBJS)        \
        $(COMSYSOBJS)           \
        $(SYSOBJS)              \
        $(COMSYSOBJS64)         \
        $(SYSOBJS64)            \
        $(VALUES)

TRACEOBJS=                      \
        plockstat.o

# NOTE: libc.so.1 must be linked with the minimal crti.o and crtn.o
# modules whose source is provided in the $(SRC)/lib/crt directory.
# This must be done because otherwise the Sun C compiler would insert
# its own versions of these modules and those versions contain code
```

```
1051 # to call out to C++ initialization functions.  Such C++ initialization
1052 # functions can call back into libc before thread initialization is
1053 # complete and this leads to segmentation violations and other problems.
1054 # Since libc contains no C++ code, linking with the minimal crti.o and
1055 # crtn.o modules is safe and avoids the problems described above.
1056 OBJECTS= $(CRTI) $(MOSTOBJS) $(CRTN)
1057 CRTSRCS= ../../crt/i386

1059 LDPASS_OFF=     $(POUND_SIGN)

1061 # include common library definitions
1062 include ../../Makefile.lib

1064 # we need to override the default SONAME here because we might
1065 # be building a variant object (still libc.so.1, but different filename)
1066 SONAME = libc.so.1

1068 CFLAGS += $(CCVERBOSE) $(CTF_FLAGS)

1070 # This is necessary to avoid problems with calling _ex_unwind().
1071 # We probably don't want any inlining anyway.
1072 XINLINE = -xinline=
1073 CFLAGS += $(XINLINE)

1075 CERRWARN += -_gcc=-Wno-parentheses
1076 CERRWARN += -_gcc=-Wno-switch
1077 CERRWARN += -_gcc=-Wno-uninitialized
1078 CERRWARN += -_gcc=-Wno-unused-value
1079 CERRWARN += -_gcc=-Wno-unused-label
1080 CERRWARN += -_gcc=-Wno-unused-variable
1081 CERRWARN += -_gcc=-Wno-type-limits
1082 CERRWARN += -_gcc=-Wno-char-subscripts
1083 CERRWARN += -_gcc=-Wno-clobbered
1084 CERRWARN += -_gcc=-Wno-unused-function
1085 CERRWARN += -_gcc=-Wno-address

1087 # not linted
1088 SMATCH=off

1090 # Setting THREAD_DEBUG = -DTHREAD_DEBUG (make THREAD_DEBUG=-DTHREAD_DEBUG ...)
1091 # enables ASSERT() checking in the threads portion of the library.
1092 # This is automatically enabled for DEBUG builds, not for non-debug builds.
1093 THREAD_DEBUG =
1094 $(NOT_RELEASE_BUILD)THREAD_DEBUG = -DTHREAD_DEBUG

1096 # Make string literals read-only to save memory.
1097 CFLAGS += $(XSTRCONST)

1099 ALTPICS= $(TRACEOBJS:%=pics/%)

1101 $(DYNLIB) := BUILD.SO = $(LD) -o $@ -G $(DYNFLAGS) $(PICS) $(ALTPICS) \
1102                 $(EXTPICS) $(LDLIBS)

1104 MAPFILES =      $(LIBCDIR)/port/mapfile-vers

1106 #
1107 # EXTN_CPPFLAGS and EXTN_CFLAGS set in enclosing Makefile
1108 #
1109 CFLAGS +=       $(EXTN_CFLAGS)
1110 CPPFLAGS=       -D_REENTRANT -Di386 $(EXTN_CPPFLAGS) $(THREAD_DEBUG) \
1111                 -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc $(CPPFLAGS.master)
1112 ASFLAGS=        $(AS_PICFLAGS) -P -D__STDC__ -D_ASM $(CPPFLAGS) $(i386_AS_XARCH)

1114 # As a favor to the dtrace syscall provider, libc still calls the
1115 # old syscall traps that have been obsoleted by the *at() interfaces.
1116 # Delete this to compile libc using only the new *at() system call traps
```



```
1117 CPPFLAGS += -D_RETAIN_OLD_SYSCALLS

1119 # Inform the run-time linker about libc specialized initialization
1120 RTLDINFO =      -z rtldinfo=tls_rtldinfo
1121 DYNFLAGS +=     $(RTLDINFO)

1123 # Force libc's internal references to be resolved immediately upon loading
1124 # in order to avoid critical region problems.  Since almost all libc symbols
1125 # are marked 'protected' in the mapfiles, this is a minimal set (15 to 20).
1126 DYNFLAGS +=     -znow

1128 DYNFLAGS +=     -e __rtboot
1129 DYNFLAGS +=     $(EXTN_DYNFLAGS)

1131 # Inform the kernel about the initial DTrace area (in case
1132 # libc is being used as the interpreter / runtime linker).
1133 DTRACE_DATA =   -zdtrace=dtrace_data
1134 DYNFLAGS +=     $(DTRACE_DATA)

1136 # DTrace needs an executable data segment.
1137 MAPFILE.NED=

1139 BUILD.s=        $(AS) $(ASFLAGS) $< -o $@

1141 # Override this top level flag so the compiler builds in its native
1142 # C99 mode.  This has been enabled to support the complex arithmetic
1143 # added to libc.
1144 CSTD=   $(CSTD_GNU99)

1146 # libc method of building an archive
1147 # The "$(GREP) -v ' L '" part is necessary only until
1148 # lorder is fixed to ignore thread-local variables.
1149 BUILD.AR= $(RM) $@ ; \
1150         $(AR) q $@ `$(LORDER) $(MOSTOBJS:%=$(DIR)/%) | $(GREP) -v ' L ' | $(TSOR

1152 # extra files for the clean target
1153 CLEANFILES +=                   \
1154         $(LIBCDIR)/port/gen/errlst.c    \
1155         $(LIBCDIR)/port/gen/new_list.c  \
1156         assym.h                 \
1157         genassym                \
1158         crt/_rtld.s             \
1159         crt/_rtbootld.s         \
1160         pics/_rtbootld.o        \
1161         pics/crti.o             \
1162         pics/crtn.o             \
1163         $(ALTPICS)

1165 CLOBBERFILES += $(LIB_PIC)

1167 # list of C source for lint
1168 SRCS=                                                   \
1169         $(ATOMICOBJS:%.o=$(SRC)/common/atomic/%.c)      \
1170         $(XATTROBJS:%.o=$(SRC)/common/xattr/%.c)        \
1171         $(COMOBJS:%.o=$(SRC)/common/util/%.c)           \
1172         $(DTRACEOBJS:%.o=$(SRC)/common/dtrace/%.c)      \
1173         $(PORTFP:%.o=$(LIBCDIR)/port/fp/%.c)            \
1174         $(PORTGEN:%.o=$(LIBCDIR)/port/gen/%.c)          \
1175         $(PORTI18N:%.o=$(LIBCDIR)/port/i18n/%.c)        \
1176         $(PORTINET:%.o=$(LIBCDIR)/port/inet/%.c)        \
1177         $(PORTLOCALE:%.o=$(LIBCDIR)/port/locale/%.c)    \
1178         $(PORTPRINT:%.o=$(LIBCDIR)/port/print/%.c)      \
1179         $(PORTREGEX:%.o=$(LIBCDIR)/port/regex/%.c)      \
1180         $(PORTSTDIO:%.o=$(LIBCDIR)/port/stdio/%.c)      \
1181         $(PORTSYS:%.o=$(LIBCDIR)/port/sys/%.c)          \
1182         $(AIOOBJS:%.o=$(LIBCDIR)/port/aio/%.c)          \
```

```
1183         $(RTOBJS:%.o=$(LIBCDIR)/port/rt/%.c)            \
1184         $(SECFLAGSOBJS:%.o=$(SRC)/common/secflags/%.c)  \
1185         $(TPOOLOBJS:%.o=$(LIBCDIR)/port/tpool/%.c)      \
1186         $(THREADSOBJS:%.o=$(LIBCDIR)/port/threads/%.c)  \
1187         $(THREADSMACHOBJS:%.o=$(LIBCDIR)/$(MACH)/threads/%.c) \
1188         $(UNICODEOBJS:%.o=$(SRC)/common/unicode/%.c)    \
1189         $(UNWINDMACHOBJS:%.o=$(LIBCDIR)/port/unwind/%.c) \
1190         $(FPOBJS:%.o=$(LIBCDIR)/$(MACH)/fp/%.c)         \
1191         $(LIBCBASE)/gen/ecvt.c                          \
1192         $(LIBCBASE)/gen/makectxt.c                      \
1193         $(LIBCBASE)/gen/siginfolst.c                    \
1194         $(LIBCBASE)/gen/siglongjmp.c                    \
1195         $(LIBCBASE)/gen/strcmp.c                        \
1196         $(LIBCBASE)/gen/sync_instruction_memory.c       \
1197         $(LIBCBASE)/sys/ptrace.c                        \
1198         $(LIBCBASE)/sys/uadmin.c

1200 # conditional assignments
1201 $(DYNLIB) := CRTI = crti.o
1202 $(DYNLIB) := CRTN = crtn.o

1204 # Files which need the threads .il inline template
1205 TIL=                            \
1206         aio.o                   \
1207         alloc.o                 \
1208         assfail.o               \
1209         atexit.o                \
1210         atfork.o                \
1211         cancel.o                \
1212         door_calls.o            \
1213         err.o                   \
1214         errno.o                 \
1215         lwp.o                   \
1216         ma.o                    \
1217         machdep.o               \
1218         posix_aio.o             \
1219         pthr_attr.o             \
1220         pthr_barrier.o          \
1221         pthr_cond.o             \
1222         pthr_mutex.o            \
1223         pthr_rwlock.o           \
1224         pthread.o               \
1225         rand.o                  \
1226         rwlock.o                \
1227         scalls.o                \
1228         sched.o                 \
1229         sema.o                  \
1230         sigaction.o             \
1231         sigev_thread.o          \
1232         spawn.o                 \
1233         stack.o                 \
1234         synch.o                 \
1235         tdb_agent.o             \
1236         thr.o                   \
1237         thread_interface.o      \
1238         thread_pool.o           \
1239         tls.o                   \
1240         tsd.o                   \
1241         tmem.o                  \
1242         unwind.o

1244 THREADS_INLINES = $(LIBCBASE)/threads/i386.il
1245 $(TIL:%=pics/%) := CFLAGS += $(THREADS_INLINES)

1247 # pics/mul64.o := CFLAGS += $(LIBCBASE)/crt/mul64.il
```



```
1249 # large-file-aware components that should be built large

1251 $(COMSYSOBJS64:%=pics/%) := \
1252         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1254 $(SYSOBJS64:%=pics/%) := \
1255         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1257 $(PORTGEN64:%=pics/%) := \
1258         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1260 $(PORTREGEX64:%=pics/%) := \
1261         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1263 $(PORTSTDIO64:%=pics/%) := \
1264         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1266 $(PORTSYS64:%=pics/%) := \
1267         CPPFLAGS += -D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64

1269 $(PORTSTDIO_W:%=pics/%) := \
1270         CPPFLAGS += -D_WIDE

1272 $(PORTPRINT_W:%=pics/%) := \
1273         CPPFLAGS += -D_WIDE

1275 $(PORTPRINT_C89:%=pics/%) := \
1276         CPPFLAGS += -D_C89_INTMAX32

1278 $(PORTSTDIO_C89:%=pics/%) := \
1279         CPPFLAGS += -D_C89_INTMAX32

1281 $(PORTI18N_COND:%=pics/%) := \
1282         CPPFLAGS += -D_WCS_LONGLONG

1284 pics/arc4random.o :=    CPPFLAGS += -I$(SRC)/common/crypto/chacha

1286 pics/__clock_gettime.o := CPPFLAGS += $(COMMPAGE_CPPFLAGS)

1288 .KEEP_STATE:

1290 all: $(LIBS) $(LIB_PIC)

1292 lint    :=      CPPFLAGS += -I$(LIBCDIR)/$(MACH)/fp
1293 lint    :=      CPPFLAGS += -D_MSE_INT_H -D_LCONV_C99
1294 lint    :=      LINTFLAGS += -mn -erroff=E_SUPPRESSION_DIRECTIVE_UNUSED

1296 lint:
1297         @echo $(LINT.c) ...
1298         @$(LINT.c) $(SRCS) $(LDLIBS)

1300 $(LINTLIB):= SRCS=$(LIBCDIR)/port/llib-lc
1301 $(LINTLIB):= CPPFLAGS += -D_MSE_INT_H
1302 $(LINTLIB):= LINTFLAGS=-nvx

1304 # object files that depend on inline template
1305 $(TIL:%=pics/%): $(LIBCBASE)/threads/i386.il
1306 # pics/mul64.o: $(LIBCBASE)/crt/mul64.il

1308 # include common libc targets
1309 include $(LIBCDIR)/Makefile.targ

1311 # We need to strip out all CTF and DOF data from the static library
1312 $(LIB_PIC) := DIR = pics
1313 $(LIB_PIC): pics $$(PICS)
1314         $(BUILD.AR)
```

```
1315         $(MCS) -d -n .SUNW_ctf $@ > /dev/null 2>&1
1316         $(MCS) -d -n .SUNW_dof $@ > /dev/null 2>&1
1317         $(AR) -ts $@ > /dev/null
1318         $(POST_PROCESS_A)

1320 $(LIBCBASE)/crt/_rtbootld.s: $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.c
1321         $(CC) $(CPPFLAGS) -_smatch=off $(CTF_FLAGS) -O -S $(C_PICFLAGS) \
1318         $(CC) $(CPPFLAGS) $(CTF_FLAGS) -O -S $(C_PICFLAGS) \
1322             $(LIBCBASE)/crt/_rtld.c -o $(LIBCBASE)/crt/_rtld.s
1323         $(CAT) $(LIBCBASE)/crt/_rtboot.s $(LIBCBASE)/crt/_rtld.s > $@
1324         $(RM) $(LIBCBASE)/crt/_rtld.s

1326 # partially built from C source
1327 pics/_rtbootld.o: $(LIBCBASE)/crt/_rtbootld.s
1328         $(AS) $(ASFLAGS) $(LIBCBASE)/crt/_rtbootld.s -o $@
1329         $(CTFCONVERT_O)

1331 ASSYMDEP_OBJS=                  \
1332         _lwp_mutex_unlock.o     \
1333         _stack_grow.o           \
1334         getcontext.o            \
1335         setjmp.o                \
1336         tls_get_addr.o          \
1337         vforkx.o

1339 $(ASSYMDEP_OBJS:%=pics/%)       :=      CPPFLAGS += -I.

1341 $(ASSYMDEP_OBJS:%=pics/%): assym.h

1343 # assym.h build rules

1345 GENASSYM_C = $(LIBCDIR)/$(MACH)/genassym.c

1347 genassym: $(GENASSYM_C)
1348         $(NATIVECC) $(NATIVE_CFLAGS) -I$(LIBCBASE)/inc -I$(LIBCDIR)/inc \
1349                 -D__EXTENSIONS__ $(CPPFLAGS.native) -o $@ $(GENASSYM_C)

1351 OFFSETS = $(LIBCDIR)/$(MACH)/offsets.in

1353 assym.h: $(OFFSETS) genassym
1354         $(OFFSETS_CREATE) <$(OFFSETS) >$@
1355         ./genassym >>$@

1357 # derived C source and related explicit dependencies
1358 $(LIBCDIR)/port/gen/errlst.c + \
1359 $(LIBCDIR)/port/gen/new_list.c: $(LIBCDIR)/port/gen/errlist $(LIBCDIR)/port/gen/
1360         cd $(LIBCDIR)/port/gen; pwd; $(AWK) -f errlist.awk < errlist

1362 pics/errlst.o: $(LIBCDIR)/port/gen/errlst.c

1364 pics/new_list.o: $(LIBCDIR)/port/gen/new_list.c
```

```
**********************************************************
    3039 Fri Dec 21 14:59:57 2018
new/usr/src/lib/libm/Makefile.libm.com
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright 2011 Nexenta Systems, Inc.  All rights reserved.
# Copyright (c) 2018, Joyent, Inc.
#

LIBMDIR         = $(SRC)/lib/libm

LIBMSRC         = $(LIBMDIR)/common

CPP_CMD         = $(CC) -E -Xs

ASSUFFIX_sparc  = S
ASSUFFIX_i386   = s
ASSUFFIX        = $(ASSUFFIX_$(MACH))

# With studio CSTD of neither enabled nor disabled is "no_lib", whereby we
# expect C99-the-language, but don't modify the behaviour of library routines.
# This is VERY IMPORTANT, as $(CSTD_GNU99), for instance, would link us with
# values-xpg6, which would introduce an __xpg6 to our object with the C99
# flags set, causing us to default C99 libm behaviour on, breaking
# compatibility.
#
# We must then, unfortunately, defeat the GNU compiler _defaulting_ to C99, by
# in that case setting it back to gnu89, which _also_ accepts C99 syntax as
# far as is important.
CSTD            =
CFLAGS          += -_gcc=-std=gnu89
CFLAGS64        += -_gcc=-std=gnu89

M4FLAGS         = -D__STDC__ -DPIC

LDBLDIR_sparc   = Q
LDBLDIR_i386    = LD
LDBLDIR         = $(LDBLDIR_$(MACH))

LM_IL           = $(LIBMDIR)/$(TARGET_ARCH)/src/locallibm.il

CFLAGS          += $(C_PICFLAGS) $(XSTRCONST) $(LM_IL)
CFLAGS64        += $(C_PICFLAGS) $(XSTRCONST) $(LM_IL)
sparc_CFLAGS    += -Wa,-xarch=v8plus

CPPFLAGS        += -I$(LIBMSRC)/C \
                -I$(LIBMSRC)/$(LDBLDIR) -I$(LIBMDIR)/$(TARGET_ARCH)/src

# GCC needs __C99FEATURES__ such that the implementations of isunordered,
# isgreaterequal, islessequal, etc, exist.  This is basically equivalent to
# providing no -xc99 to Studio, in that it gets us the C99 language features,
# but not values-xpg6, the reason for which is outlined with CSTD.
CFLAGS          += -_gcc=-D__C99FEATURES__
```



```
CFLAGS64        += -_gcc=-D__C99FEATURES__

# libm depends on integer overflow characteristics
CFLAGS          += -_gcc=-fno-strict-overflow
CFLAGS64        += -_gcc=-fno-strict-overflow

# sparse currently has no _Complex support
CFLAGS          += -_smatch=off
CFLAGS64        += -_smatch=off

$(DYNLIB)       := LDLIBS += -lc

$(LINTLIB)      := SRCS = $(LIBMSRC)/$(LINTSRC)

CLEANFILES      += pics/*.s pics/*.S

FPDEF_amd64     = -DARCH_amd64
FPDEF_sparc     = -DCG89 -DARCH_v8plus -DFPADD_TRAPS_INCOMPLETE_ON_NAN
FPDEF_sparcv9   = -DARCH_v9 -DFPADD_TRAPS_INCOMPLETE_ON_NAN
FPDEF           = $(FPDEF_$(TARGET_ARCH))

ASFLAGS         = -P -D_ASM $(FPDEF)

XARCH_sparc     = v8plus
XARCH_sparcv9   = v9
XARCH_i386      = f80387
XARCH_amd64     = amd64
XARCH           = $(XARCH_$(TARGET_ARCH))

ASOPT_sparc     = -xarch=$(XARCH) $(AS_PICFLAGS)
ASOPT_sparcv9   = -xarch=$(XARCH) $(AS_PICFLAGS)
ASOPT_i386      =
ASOPT_amd64     = -xarch=$(XARCH) $(AS_PICFLAGS)
ASOPT           = $(ASOPT_$(TARGET_ARCH))

ASFLAGS         += $(ASOPT)

CPPFLAGS_sparc = -DFPADD_TRAPS_INCOMPLETE_ON_NAN \
        -DFDTOS_TRAPS_INCOMPLETE_IN_FNS_MODE

CPPFLAGS        += $(CPPFLAGS_$(MACH))
ASFLAGS         += $(CPPFLAGS)
```

```
**********************************************************
    3759 Fri Dec 21 14:59:57 2018
new/usr/src/lib/libpp/Makefile.com
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.

SHELL=/usr/bin/ksh93

LIBRARY=        libpp.a
VERS=           .1

OBJECTS= \
        ppargs.o \
        ppbuiltin.o \
        ppcall.o \
        ppcomment.o \
        ppcontext.o \
        ppcontrol.o \
        ppcpp.o \
        ppdata.o \
        pperror.o \
        ppexpr.o \
        ppfsm.o \
        ppincref.o \
        ppinput.o \
        ppkey.o \
        pplex.o \
        ppline.o \
        ppmacref.o \
        ppmisc.o \
        ppop.o \
        pppragma.o \
        ppprintf.o \
        ppproto.o \
        ppsearch.o \
        pptrace.o

include ../../Makefile.astmsg

include ../../Makefile.lib
```



```
# mapfile-vers does not live with the sources in in common/ to make
# automated code updates easier.
MAPFILES=       ../mapfile-vers

# Set common AST build flags (e.g. C99/XPG6, needed to support the math stuff)
include ../../../Makefile.ast

LIBS =          $(DYNLIB) $(LINTLIB)

LDLIBS += \
        -last \
        -lc

$(LINTLIB) :=   SRCS = $(SRCDIR)/$(LINTSRC)

SRCDIR =        ../common

# We use "=" here since using $(CPPFLAGS.master) is very tricky in our
# case - it MUST come as the last element but future changes in -D options
# may then cause silent breakage in the AST sources because the last -D
# option specified overrides previous -D options so we prefer the current
# way to explicitly list each single flag.
CPPFLAGS = \
        $(DTEXTDOM) $(DTS_ERRNO) \
        -I. \
        -I$(ROOT)/usr/include/ast \
        -I$(ROOT)/usr/include \
        -D_PACKAGE_ast \
        '-DUSAGE_LICENSE=\
                "[-author?Glenn Fowler <gsf@research.att.com>]"\
                "[-copyright?Copyright (c) 1986-2009 AT&T Intellectual Property]
                "[-license?http://www.opensource.org/licenses/cpl1.0.txt]"\
                "[--catalog?libpp]"'


CFLAGS += \
        $(ASTCFLAGS)
CFLAGS64 += \
        $(ASTCFLAGS64)

CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-char-subscripts
CERRWARN        += -_gcc=-Wno-empty-body
CERRWARN        += -_gcc=-Wno-unused-value

# "pplex() parse error: turning off implications after 60 seconds"
SMATCH          = off

pics/ppcall.o           := CERRWARN += -erroff=E_INTEGER_OVERFLOW_DETECTED
pics/ppcontrol.o        := CERRWARN += -erroff=E_INTEGER_OVERFLOW_DETECTED
pics/ppcpp.o            := CERRWARN += -erroff=E_INTEGER_OVERFLOW_DETECTED
pics/ppexpr.o           := CERRWARN += -erroff=E_INTEGER_OVERFLOW_DETECTED
pics/pplex.o            := CERRWARN += -erroff=E_INTEGER_OVERFLOW_DETECTED
pics/ppop.o             := CERRWARN += -erroff=E_INTEGER_OVERFLOW_DETECTED
pics/ppsearch.o         := CERRWARN += -erroff=E_INTEGER_OVERFLOW_DETECTED
pics/ppsearch.o         := CERRWARN += -_gcc=-Wno-sequence-point
pics/pplex.o            := CERRWARN += -_gcc=-Wno-implicit-fallthrough
pics/ppcpp.o            := CERRWARN += -_gcc=-Wno-implicit-fallthrough
pics/ppproto.o          := CERRWARN += -_gcc=-Wno-implicit-fallthrough

.KEEP_STATE:

all: $(LIBS)
```

```
127 #
128 # libpp is not lint-clean yet; fake up a target.  (You can use
129 # "make lintcheck" to actually run lint; please send all lint fixes
130 # upstream (to AT&T) so the next update will pull them into ON.)
131 #
132 lint:
133         @ print "usr/src/lib/libpp is not lint-clean: skipping"

135 include ../../Makefile.targ
```

```
**********************************************************
    4379 Fri Dec 21 14:59:57 2018
new/usr/src/lib/libshell/Makefile.com
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 2007, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2018, Joyent, Inc.
#

SHELL=/usr/bin/ksh93

LIBRARY=        libshell.a
VERS=           .1

OBJECTS= \
        bltins/alarm.o \
        bltins/cd_pwd.o \
        bltins/cflow.o \
        bltins/enum.o \
        bltins/getopts.o \
        bltins/hist.o \
        bltins/misc.o \
        bltins/poll_solaris.o \
        bltins/print.o \
        bltins/read.o \
        bltins/regress.o \
        bltins/shiocmd_solaris.o \
        bltins/sleep.o \
        bltins/test.o \
        bltins/trap.o \
        bltins/typeset.o \
        bltins/ulimit.o \
        bltins/umask.o \
        bltins/whence.o \
        data/aliases.o \
        data/builtins.o \
        data/keywords.o \
        data/lexstates.o \
        data/limits.o \
        data/msg.o \
        data/options.o \
        data/signals.o \
        data/strdata.o \
```



```
        data/testops.o \
        data/variables.o \
        edit/completion.o \
        edit/edit.o \
        edit/emacs.o \
        edit/hexpand.o \
        edit/history.o \
        edit/vi.o \
        sh/args.o \
        sh/arith.o \
        sh/array.o \
        sh/defs.o \
        sh/deparse.o \
        sh/expand.o \
        sh/fault.o \
        sh/fcin.o \
        sh/init.o \
        sh/io.o \
        sh/jobs.o \
        sh/lex.o \
        sh/macro.o \
        sh/main.o \
        sh/name.o \
        sh/nvdisc.o \
        sh/nvtree.o \
        sh/nvtype.o \
        sh/parse.o \
        sh/path.o \
        sh/streval.o \
        sh/string.o \
        sh/subshell.o \
        sh/tdump.o \
        sh/timers.o \
        sh/trestore.o \
        sh/waitevent.o \
        sh/xec.o

# We are storing the object files into subdirs avoid the
# confusion with having too many object files in the toplevel pics/
# directory (this matches the way how the original AST build system
# deals with this "logistic" issue) - the rules below ensure that
# the destination directory is available.
OBJDIRS =  \
        bltins \
        data \
        edit \
        sh
PICSDIRS= $(OBJDIRS:%=pics/%)
mkpicdirs:
        @mkdir -p $(PICSDIRS)

# Specify the MACH we currently use to build and test ksh
LIBSHELLMACH= $(TARGETMACH)
LIBSHELLBASE=..

include ../../Makefile.astmsg

include ../../Makefile.lib

# mapfile-vers does not live with the sources in in common/ to make
# automated code updates easier.
MAPFILES=       ../mapfile-vers

# Set common AST build flags (e.g. C99/XPG6, needed to support the math stuff)
include ../../../Makefile.ast
```

```
127 LIBS =          $(DYNLIB) $(LINTLIB)

129 LDLIBS += \
130         -lcmd \
131         -ldll \
132         -last \
133         -lsocket \
134         -lm \
135         -lc

137 $(LINTLIB) :=   SRCS = $(SRCDIR)/$(LINTSRC)

139 SRCDIR =        ../common

141 # 1. Make sure that the -D/-U defines in CPPFLAGS below are in sync
142 # with usr/src/cmd/ksh/Makefile.com
143 # 2. We use "=" here since using $(CPPFLAGS.master) is very tricky in our
144 # case - it MUST come as the last element but future changes in -D options
145 # may then cause silent breakage in the AST sources because the last -D
146 # option specified overrides previous -D options so we prefer the current
147 # way to explicitly list each single flag.
148 CPPFLAGS = \
149         $(DTEXTDOM) $(DTS_ERRNO) \
150         $(LIBSHELLCPPFLAGS)

152 CFLAGS += \
153         $(ASTCFLAGS)
154 CFLAGS64 += \
155         $(ASTCFLAGS64)

157 CERRWARN        += -_gcc=-Wno-parentheses
158 CERRWARN        += -_gcc=-Wno-unused-value
159 CERRWARN        += -_gcc=-Wno-unused-variable
160 CERRWARN        += -_gcc=-Wno-unused-function
161 CERRWARN        += -_gcc=-Wno-uninitialized
162 CERRWARN        += -_gcc=-Wno-clobbered
163 CERRWARN        += -_gcc=-Wno-char-subscripts

165 # smatch gets out of memory on common/sh/macro.c
166 SMATCH          = off

168 pics/sh/macro.o         := CERRWARN += -erroff=E_NO_IMPLICIT_DECL_ALLOWED
169 pics/sh/nvdisc.o        := CERRWARN += -erroff=E_END_OF_LOOP_CODE_NOT_REACHED

171 .KEEP_STATE:

173 all: mkpicdirs .WAIT $(LIBS)

175 #
176 # libshell is not lint-clean yet; fake up a target.  (You can use
177 # "make lintcheck" to actually run lint; please send all lint fixes
178 # upstream (to AT&T) so the next update will pull them into ON.)
179 #
180 lint:
181         @ print "usr/src/lib/libshell is not lint-clean: skipping"

183 include ../../Makefile.targ
```

```
**********************************************************
    3621 Fri Dec 21 14:59:57 2018
new/usr/src/tools/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1999, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2014 Garrett D'Amore <garrett@damore.org>
# Copyright 2016 Toomas Soome <tsoome@me.com>
# Copyright (c) 2016, Chris Fraire <cfraire@me.com>.
# Copyright (c) 2018, Joyent, Inc.
#

include ../Makefile.master

SMATCH_1 = smatch

# Bootstrap problem: 'cw' must be built before anything else can be built.
# 'install.bin' should be built next, being the 'install' target dependency
# for everything else.
#
# Because of somewhat cyclic dependency between them, both cw and install.bin
# override the way we install binaries in their Makefiles.
BOOT_SUBDIRS= \
        $(SMATCH_$(ENABLE_SMATCH)) \
        .WAIT \
        cw \
        .WAIT \
        install.bin \
        .WAIT \
        ctf

COMMON_SUBDIRS= \
        codereview \
        codesign \
        cscope-fast \
        env \
        findunref \
        lintdump \
        make \
        makesoftcore \
        ndrgen \
        onbld \
        protocmp \
```



```
        protolist \
        scripts

#
#  special versions of commands for use only in build
#
UNSHIPPED_SUBDIRS = \
        localedef \
        mandoc \
        tic \
        vtfontcvt \
        zic

sparc_SUBDIRS= \
        chk4ubin \
        stabs \
        tokenize

i386_SUBDIRS=           \
        aw              \
        elfextract      \
        mbh_patch       \
        btxld

LINTSUBDIRS= \
        codereview \
        ctf \
        cw \
        findunref \
        lintdump \
        ndrgen \
        protocmp \
        protolist

SUBDIRS= \
        $($(MACH)_SUBDIRS) \
        $(COMMON_SUBDIRS) \
        $(UNSHIPPED_SUBDIRS)

include Makefile.tools

ROOTDIRS= \
        $(ROOTOPT) \
        $(ROOTONBLD) \
        $(ROOTONBLD)/bin \
        $(ROOTONBLD)/bin/$(MACH) \
        $(ROOTONBLD)/lib \
        $(ROOTONBLD)/lib/$(MACH) \
        $(ROOTONBLD)/lib/$(MACH)/64 \
        $(ROOTONBLD)/lib/perl \
        $(ROOTONBLD)/lib/python$(PYTHON_VERSION) \
        $(ROOTONBLD)/lib/python$(PYTHON_VERSION)/onbld \
        $(ROOTONBLD)/lib/python$(PYTHON_VERSION)/onbld/Checks \
        $(ROOTONBLD)/lib/python$(PYTHON_VERSION)/onbld/Scm \
        $(ROOTONBLD)/env \
        $(ROOTONBLD)/etc \
        $(ROOTONBLD)/etc/exception_lists \
        $(ROOTONBLD)/share \
        $(ROOTONBLD)/man \
        $(ROOTONBLD)/man/man1onbld

all :=          TARGET= install
install :=      TARGET= install
clean :=        TARGET= clean
clobber :=      TARGET= clobber
lint :=         TARGET= lint
```

```
 127 _msg :=         TARGET= _msg

 129 .KEEP_STATE:

 131 #
 132 # Only create directories in the tools proto area when doing an actual
 133 # build, not a clean or clobber.
 134 #
 135 DOROOTDIRS= $(ROOTDIRS)
 136 clobber:= DOROOTDIRS=
 137 clean:= DOROOTDIRS=

 139 DOROOTONBLDLIBPY= $(ROOTONBLDLIBPY)
 140 clobber:= DOROOTONBLDLIBPY=
 141 clean:= DOROOTONBLDLIBPY=

 143 all install: $(SUBDIRS)

 145 clean: $(SUBDIRS)

 147 clobber: $(SUBDIRS)
 148         $(RM) -rf $(TOOLS_PROTO)

 150 lint: $(LINTSUBDIRS)

 152 _msg: $(MSGSUBDIRS)

 154 .PARALLEL: $(SUBDIRS) $(CLOSED_SUBDIRS)

 156 $(SUBDIRS) $(CLOSED_SUBDIRS): $(BOOT_SUBDIRS)

 158 $(BOOT_SUBDIRS) $(SUBDIRS): $$(DOROOTDIRS) $$(DOROOTONBLDLIBPY) FRC
 159         @cd $@; pwd; $(MAKE) $(TARGET)

 161 # Assume we don't have the install.bin available yet
 162 $(ROOTDIRS):
 163         $(MKDIR) -p -m $(DIRMODE) $@

 165 $(ROOTONBLDLIBPY): $(ROOTDIRS)
 166         $(RM) -r $@; $(SYMLINK) python$(PYTHON_VERSION) $@

 168 FRC:
```

```
**********************************************************
   41239 Fri Dec 21 14:59:58 2018
new/usr/src/tools/cw/cw.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************

/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2018, Richard Lowe.
 */
/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 *
 * Copyright 2018 Joyent, Inc.
 */

/*
 * Wrapper for the GNU C compiler to make it accept the Sun C compiler
 * arguments where possible.
 *
 * Since the translation is inexact, this is something of a work-in-progress.
 *
 */

/* If you modify this file, you must increment CW_VERSION */
#define CW_VERSION      "3.0"

/*
 * -#           Verbose mode
 * -###         Show compiler commands built by driver, no compilation
 * -A<name[(tokens)]>   Preprocessor predicate assertion
 * -B<[static|dynamic]> Specify dynamic or static binding
 * -C           Prevent preprocessor from removing comments
 * -c           Compile only - produce .o files, suppress linking
 * -cg92        Alias for -xtarget=ss1000
 * -D<name[=token]>     Associate name with token as if by #define
 * -d[y|n]      dynamic [-dy] or static [-dn] option to linker
 * -E           Compile source through preprocessor only, output to stdout
 * -erroff=<t>  Suppress warnings specified by tags t(%none, %all, <tag list>)
 * -errtags=<a> Display messages with tags a(no, yes)
 * -errwarn=<t> Treats warnings specified by tags t(%none, %all, <tag list>)
 *              as errors
 * -fast        Optimize using a selection of options
 * -fd          Report old-style function definitions and declarations
```



```
 * -fnonstd     Initialize floating-point hardware to non-standard preferences
 * -fns[=<yes|no>] Select non-standard floating point mode
 * -fprecision=<p> Set FP rounding precision mode p(single, double, extended)
 * -fround=<r>  Select the IEEE rounding mode in effect at startup
 * -fsimple[=<n>] Select floating-point optimization preferences <n>
 * -fsingle     Use single-precision arithmetic (-Xt and -Xs modes only)
 * -ftrap=<t>   Select floating-point trapping mode in effect at startup
 * -fstore      force floating pt. values to target precision on assignment
 * -G           Build a dynamic shared library
 * -g           Compile for debugging
 * -H           Print path name of each file included during compilation
 * -h <name>    Assign <name> to generated dynamic shared library
 * -I<dir>      Add <dir> to preprocessor #include file search path
 * -i           Passed to linker to ignore any LD_LIBRARY_PATH setting
 * -keeptmp     Keep temporary files created during compilation
 * -L<dir>      Pass to linker to add <dir> to the library search path
 * -l<name>     Link with library lib<name>.a or lib<name>.so
 * -mc          Remove duplicate strings from .comment section of output files
 * -mr          Remove all strings from .comment section of output files
 * -mr,"string" Remove all strings and append "string" to .comment section
 * -mt          Specify options needed when compiling multi-threaded code
 * -native      Find available processor, generate code accordingly
 * -nofstore    Do not force floating pt. values to target precision
 *              on assignment
 * -norunpath   Do not build in a runtime path for shared libraries
 * -O           Use default optimization level (-xO2 or -xO3. Check man page.)
 * -o <outputfile> Set name of output file to <outputfile>
 * -P           Compile source through preprocessor only, output to .i  file
 * -p           Compile for profiling with prof
 * -Q[y|n]      Emit/don't emit identification info to output file
 * -R<dir[:dir]> Build runtime search path list into executable
 * -S           Compile and only generate assembly code (.s)
 * -s           Strip symbol table from the executable file
 * -t           Turn off duplicate symbol warnings when linking
 * -U<name>     Delete initial definition of preprocessor symbol <name>
 * -V           Report version number of each compilation phase
 * -v           Do stricter semantic checking
 * -W<c>,<arg>  Pass <arg> to specified component <c> (a,l,m,p,0,2,h,i,u)
 * -w           Suppress compiler warning messages
 * -Xa          Compile assuming ANSI C conformance, allow K & R extensions
 *              (default mode)
 * -Xs          Compile assuming (pre-ANSI) K & R C style code
 * -Xt          Compile assuming K & R conformance, allow ANSI C
 * -xarch=<a>   Specify target architecture instruction set
 * -xbuiltin[=<b>] When profitable inline, or substitute intrinisic functions
 *              for system functions, b={%all,%none}
 * -xCC         Accept C++ style comments
 * -xchip=<c>   Specify the target processor for use by the optimizer
 * -xcode=<c>   Generate different code for forming addresses
 * -xcrossfile[=<n>] Enable optimization and inlining across source files,
 *              n={0|1}
 * -xe          Perform only syntax/semantic checking, no code generation
 * -xF          Compile for later mapfile reordering or unused section
 *              elimination
 * -xhelp=<f>   Display on-line help information f(flags, readme, errors)
 * -xildoff     Cancel -xildon
 * -xildon      Enable use of the incremental linker, ild
 * -xinline=[<a>,...,<a>]  Attempt inlining of specified user routines,
 *              <a>={%auto,func,no%func}
 * -xlibmieee   Force IEEE 754 return values for math routines in
 *              exceptional cases
 * -xlibmil     Inline selected libm math routines for optimization
 * -xlic_lib=sunperf    Link in the Sun supplied performance libraries
 * -xlicinfo    Show license server information
 * -xmaxopt=[off,1,2,3,4,5] maximum optimization level allowed on #pragma opt
 * -xO<n>       Generate optimized code (n={1|2|3|4|5})
```

```
 * -xP          Print prototypes for function definitions
 * -xprofile=<p> Collect data for a profile or use a profile to optimize
 *              <p>={{collect,use}[:<path>],tcov}
 * -xregs=<r>   Control register allocation
 * -xs          Allow debugging without object (.o) files
 * -xsb         Compile for use with the WorkShop source browser
 * -xsbfast     Generate only WorkShop source browser info, no compilation
 * -xsfpconst   Represent unsuffixed floating point constants as single
 *              precision
 * -xspace      Do not do optimizations that increase code size
 * -xstrconst   Place string literals into read-only data segment
 * -xtarget=<t> Specify target system for optimization
 * -xtemp=<dir> Set directory for temporary files to <dir>
 * -xtime       Report the execution time for each compilation phase
 * -xunroll=n   Enable unrolling loops n times where possible
 * -Y<c>,<dir>  Specify <dir> for location of component <c> (a,l,m,p,0,h,i,u)
 * -YA,<dir>    Change default directory searched for components
 * -YI,<dir>    Change default directory searched for include files
 * -YP,<dir>    Change default directory for finding libraries files
 * -YS,<dir>    Change default directory for startup object files
 */

/*
 * Translation table:
 */
/*
 * -#                           -v
 * -###                         error
 * -A<name[(tokens)]>           pass-thru
 * -B<[static|dynamic]>         pass-thru (syntax error for anything else)
 * -C                           pass-thru
 * -c                           pass-thru
 * -cg92                        -m32 -mcpu=v8 -mtune=supersparc (SPARC only)
 * -D<name[=token]>             pass-thru
 * -dy or -dn                   -Wl,-dy or -Wl,-dn
 * -E                           pass-thru
 * -erroff=E_EMPTY_TRANSLATION_UNIT ignore
 * -errtags=%all                -Wall
 * -errwarn=%all                -Werror else -Wno-error
 * -fast                        error
 * -fd                          error
 * -fnonstd                     error
 * -fns[=<yes|no>]              error
 * -fprecision=<p>              error
 * -fround=<r>                  error
 * -fsimple[=<n>]               error
 * -fsingle[=<n>]               error
 * -ftrap=<t>                   error
 * -fstore                      error
 * -G                           pass-thru
 * -g                           pass-thru
 * -H                           pass-thru
 * -h <name>                    pass-thru
 * -I<dir>                      pass-thru
 * -i                           pass-thru
 * -keeptmp                     -save-temps
 * -L<dir>                      pass-thru
 * -l<name>                     pass-thru
 * -mc                          error
 * -mr                          error
 * -mr,"string"                 error
 * -mt                          -D_REENTRANT
 * -native                      error
 * -nofstore                    error
 * -nolib                       -nodefaultlibs
 * -norunpath                   ignore
```



```
 * -O                           -O1 (Check the man page to be certain)
 * -o <outputfile>              pass-thru
 * -P                           -E -o filename.i (or error)
 * -p                           pass-thru
 * -Q[y|n]                      error
 * -R<dir[:dir]>                pass-thru
 * -S                           pass-thru
 * -s                           -Wl,-s
 * -t                           -Wl,-t
 * -U<name>                     pass-thru
 * -V                           --version
 * -v                           -Wall
 * -Wa,<arg>                    pass-thru
 * -Wp,<arg>                    pass-thru except -xc99=<a>
 * -Wl,<arg>                    pass-thru
 * -W{m,0,2,h,i,u>              error/ignore
 * -xmodel=kernel               -ffreestanding -mcmodel=kernel -mno-red-zone
 * -Wu,-save_args               -msave-args
 * -w                           pass-thru
 * -Xa                          -std=iso9899:199409 or -ansi
 * -Xt                          error
 * -Xs                          -traditional -std=c89
 * -xarch=<a>                   table
 * -xbuiltin[=<b>]              -fbuiltin (-fno-builtin otherwise)
 * -xCC                         ignore
 * -xchip=<c>                   table
 * -xcode=<c>                   table
 * -xdebugformat=<format>       ignore (always use dwarf-2 for gcc)
 * -xcrossfile[=<n>]            ignore
 * -xe                          error
 * -xF                          error
 * -xhelp=<f>                   error
 * -xildoff                     ignore
 * -xildon                      ignore
 * -xinline                     ignore
 * -xlibmieee                   error
 * -xlibmil                     error
 * -xlic_lib=sunperf            error
 * -xmaxopt=[...]               error
 * -xO<n>                       -O<n>
 * -xP                          error
 * -xprofile=<p>                error
 * -xregs=<r>                   table
 * -xs                          error
 * -xsb                         error
 * -xsbfast                     error
 * -xsfpconst                   error
 * -xspace                      ignore (-not -Os)
 * -xstrconst                   ignore
 * -xtarget=<t>                 table
 * -xtemp=<dir>                 error
 * -xtime                       error
 * -xtransition                 -Wtransition
 * -xunroll=n                   error
 * -W0,-xdbggen=no%usedonly     -fno-eliminate-unused-debug-symbols
 *                              -fno-eliminate-unused-debug-types
 * -Y<c>,<dir>                  error
 * -YA,<dir>                    error
 * -YI,<dir>                    -nostdinc -I<dir>
 * -YP,<dir>                    error
 * -YS,<dir>                    error
 */

#include <ctype.h>
#include <err.h>
#include <errno.h>
```

```
 259 #include <fcntl.h>
 260 #include <getopt.h>
 261 #include <stdio.h>
 262 #include <stdlib.h>
 263 #include <string.h>
 264 #include <unistd.h>

 266 #include <sys/param.h>
 267 #include <sys/stat.h>
 268 #include <sys/types.h>
 269 #include <sys/utsname.h>
 270 #include <sys/wait.h>

 272 #define CW_F_CXX        0x01
 273 #define CW_F_SHADOW     0x02
 274 #define CW_F_EXEC       0x04
 275 #define CW_F_ECHO       0x08
 276 #define CW_F_XLATE      0x10
 277 #define CW_F_PROG       0x20

 279 typedef enum cw_op {
 280         CW_O_NONE = 0,
 281         CW_O_PREPROCESS,
 282         CW_O_COMPILE,
 283         CW_O_LINK
 284 } cw_op_t;
______unchanged_portion_omitted_

 294 typedef enum {
 295         GNU,
 296         SUN,
 297         SMATCH
 294         SUN
 298 } compiler_style_t;
______unchanged_portion_omitted_

1276 static void
1277 do_smatch(cw_ictx_t *ctx)
1278 {
1279         if (ctx->i_flags & CW_F_PROG) {
1280                 newae(ctx->i_ae, "--version");
1281                 return;
1282         }

1284         /*
1285          * Some sources shouldn't run smatch at all.
1286          */
1287         for (int i = 0; i < ctx->i_oldargc; i++) {
1288                 char *arg = ctx->i_oldargv[i];

1290                 if (strcmp(arg, "-_smatch=off") == 0) {
1291                         ctx->i_flags &= ~ (CW_F_EXEC | CW_F_ECHO);
1292                         return;
1293                 }
1294         }

1296         /*
1297          * smatch can handle gcc's options.
1298          */
1299         do_gcc(ctx);
1300 }

1302 static void
1303 do_cc(cw_ictx_t *ctx)
1304 {
1305         int in_output = 0, seen_o = 0;
```



```
1306         cw_op_t op = CW_O_LINK;
1307         char *nameflag;

1309         if (ctx->i_flags & CW_F_PROG) {
1310                 newae(ctx->i_ae, "-V");
1311                 return;
1312         }

1314         if (asprintf(&nameflag, "-_%s=", ctx->i_compiler->c_name) == -1)
1315                 nomem();

1317         while (--ctx->i_oldargc > 0) {
1318                 char *arg = *++ctx->i_oldargv;

1320                 if (strncmp(arg, "-_CC=", 5) == 0) {
1321                         newae(ctx->i_ae, strchr(arg, '=') + 1);
1322                         continue;
1323                 }

1325                 if (*arg != '-') {
1326                         if (in_output == 0 || !(ctx->i_flags & CW_F_SHADOW)) {
1327                                 newae(ctx->i_ae, arg);
1328                         } else {
1329                                 in_output = 0;
1330                                 newae(ctx->i_ae, ctx->i_discard);
1331                         }
1332                         continue;
1333                 }
1334                 switch (*(arg + 1)) {
1335                 case '_':
1336                         if ((strncmp(arg, nameflag, strlen(nameflag)) == 0) ||
1337                             (strncmp(arg, "-_cc=", 5) == 0) ||
1338                             (strncmp(arg, "-_sun=", 6) == 0)) {
1339                                 newae(ctx->i_ae, strchr(arg, '=') + 1);
1340                         }
1341                         break;

1343                 case 'V':
1344                         ctx->i_flags &= ~CW_F_ECHO;
1345                         newae(ctx->i_ae, arg);
1346                         break;
1347                 case 'o':
1348                         seen_o = 1;
1349                         if (strlen(arg) == 2) {
1350                                 in_output = 1;
1351                                 newae(ctx->i_ae, arg);
1352                         } else if (ctx->i_flags & CW_F_SHADOW) {
1353                                 newae(ctx->i_ae, "-o");
1354                                 newae(ctx->i_ae, ctx->i_discard);
1355                         } else {
1356                                 newae(ctx->i_ae, arg);
1357                         }
1358                         break;
1359                 case 'c':
1360                 case 'S':
1361                         if (strlen(arg) == 2)
1362                                 op = CW_O_COMPILE;
1363                         newae(ctx->i_ae, arg);
1364                         break;
1365                 case 'E':
1366                 case 'P':
1367                         if (strlen(arg) == 2)
1368                                 op = CW_O_PREPROCESS;
1369                 /*FALLTHROUGH*/
1370                 default:
1371                         newae(ctx->i_ae, arg);
```

```
1372                 }
1373         }

1375         free(nameflag);

1377         if ((op == CW_O_LINK || op == CW_O_PREPROCESS) &&
1378             (ctx->i_flags & CW_F_SHADOW))
1379                 exit(0);

1381         if (!seen_o && (ctx->i_flags & CW_F_SHADOW)) {
1382                 newae(ctx->i_ae, "-o");
1383                 newae(ctx->i_ae, ctx->i_discard);
1384         }
1385 }

1387 static void
1388 prepctx(cw_ictx_t *ctx)
1389 {
1390         newae(ctx->i_ae, ctx->i_compiler->c_path);

1392         if (ctx->i_flags & CW_F_PROG) {
1393                 (void) printf("%s: %s\n", (ctx->i_flags & CW_F_SHADOW) ?
1394                     "shadow" : "primary", ctx->i_compiler->c_path);
1395                 (void) fflush(stdout);
1396         }

1398         if (!(ctx->i_flags & CW_F_XLATE))
1399                 return;

1401         switch (ctx->i_compiler->c_style) {
1402         case SUN:
1403                 do_cc(ctx);
1404                 break;
1405         case GNU:
1406                 do_gcc(ctx);
1407                 break;
1408         case SMATCH:
1409                 do_smatch(ctx);
1410                 break;
1411         }
1412 }
```

*unchanged_portion_omitted_*

```
1575 static void
1576 parse_compiler(const char *spec, cw_compiler_t *compiler)
1577 {
1578         char *tspec, *token;

1580         if ((tspec = strdup(spec)) == NULL)
1581                 nomem();

1583         if ((token = strsep(&tspec, ",")) == NULL)
1584                 errx(1, "Compiler is missing a name: %s", spec);
1585         compiler->c_name = token;

1587         if ((token = strsep(&tspec, ",")) == NULL)
1588                 errx(1, "Compiler is missing a path: %s", spec);
1589         compiler->c_path = token;

1591         if ((token = strsep(&tspec, ",")) == NULL)
1592                 errx(1, "Compiler is missing a style: %s", spec);

1594         if ((strcasecmp(token, "gnu") == 0) ||
1595             (strcasecmp(token, "gcc") == 0)) {
1563             (strcasecmp(token, "gcc") == 0))
1596                 compiler->c_style = GNU;
```



```
1597         } else if ((strcasecmp(token, "sun") == 0) ||
1598             (strcasecmp(token, "cc") == 0)) {
1565         else if ((strcasecmp(token, "sun") == 0) ||
1566             (strcasecmp(token, "cc") == 0))
1599                 compiler->c_style = SUN;
1600         } else if ((strcasecmp(token, "smatch") == 0)) {
1601                 compiler->c_style = SMATCH;
1602         } else {
1568         else
1603                 errx(1, "unknown compiler style: %s", token);
1604         }

1606         if (tspec != NULL)
1607                 errx(1, "Excess tokens in compiler: %s", spec);
1608 }
```

*unchanged_portion_omitted_*

```
**********************************************************
   10616 Fri Dec 21 14:59:58 2018
new/usr/src/tools/env/illumos.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
# Copyright (c) 2005, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright 2015 Nexenta Systems, Inc.  All rights reserved.
# Copyright 2012 Joshua M. Clulow <josh@sysmgr.org>
# Copyright 2015, OmniTI Computer Consulting, Inc. All rights reserved.
# Copyright 2018 OmniOS Community Edition (OmniOSce) Association.
# Copyright (c) 2018, Joyent, Inc.
#
# - This file is sourced by "bldenv.sh" and "nightly.sh" and should not
#   be executed directly.
# - This script is only interpreted by ksh93 and explicitly allows the
#   use of ksh93 language extensions.


# ------------------------------------------------------------------------------
# Parameters you are likely to want to change
# ------------------------------------------------------------------------------

#       DEBUG build only (-D, -F)
#       do not bringover from the parent (-n)
#       runs 'make check' (-C)
#       checks for new interfaces in libraries (-A)
#       sends mail on completion (-m and the MAILTO variable)
#       creates packages for PIT/RE (-p)
#       checks for changes in ELF runpaths (-r)
#       build and use this workspace's tools in $SRC/tools (-t)
export NIGHTLY_OPTIONS='-FnCDAmprt'

# Some scripts optionally send mail messages to MAILTO.
#export MAILTO=

# CODEMGR_WS - where is your workspace at
export CODEMGR_WS="'git rev-parse --show-toplevel'"

# Compilers may be specified using the following variables:
# PRIMARY_CC    - primary C compiler
# PRIMARY_CCC   - primary C++ compiler
#
# SHADOW_CCS    - list of shadow C compilers
# SHADOW_CCCS   - list of shadow C++ compilers
#
```



```
# Each entry has the form <name>,<path to binary>,<style> where name is a
# free-form name (possibly used in the makefiles to guard options), path is
# the path to the executable.  style is the 'style' of command line taken by
# the compiler, currently either gnu (or gcc) or sun (or cc), which is also
# used by Makefiles to guard options.
#
# __SUNC and __GNUC must still be set to reflect the style of the primary
# compiler (and to influence the default primary, otherwise)
#
# for example:
# export PRIMARY_CC=gcc4,/opt/gcc/4.4.4/bin/gcc,gnu
# export PRIMARY_CCC=gcc4,/opt/gcc/4.4.4/bin/g++,gnu
# export SHADOW_CCS=studio12,/opt/SUNWspro/bin/cc,sun
# export SHADOW_CCCS=studio12,/opt/SUNWspro/bin/CC,sun
#
# There can be several space-separated entries in SHADOW_* to run multiple
# shadow compilers.
#
# To disable shadow compilation, unset SHADOW_* or set them to the empty string.
#
export SHADOW_CCS=gcc7,/usr/gcc/7/bin/gcc,gnu
export SHADOW_CCCS=gcc7,/usr/gcc/7/bin/g++,gnu

# uncomment to enable smatch
#export ENABLE_SMATCH=1

# Comment this out to disable support for SMB printing, i.e. if you
# don't want to bother providing the CUPS headers this needs.
export ENABLE_SMB_PRINTING=

# If your distro uses certain versions of Perl, make sure either Makefile.master
# contains your new defaults OR your .env file sets them.
# These are how you would override for building on OmniOS r151028, for example.
#export PERL_VERSION=5.28
#export PERL_ARCH=i86pc-solaris-thread-multi-64int
#export PERL_PKGVERS=

# If your distro uses certain versions of Python, make sure either
# Makefile.master contains your new defaults OR your .env file sets them.
#export PYTHON_VERSION=2.7
#export PYTHON_PKGVERS=-27
#export PYTHON_SUFFIX=
#export PYTHON3_VERSION=3.5
#export PYTHON3_PKGVERS=-35
#export PYTHON3_SUFFIX=m

# To disable building with either Python2 or Python 3 (or both), uncomment
# these lines:
#export BUILDPY2='#'
#export BUILDPY3='#'

# Set if your distribution has different package versioning
#export PKGVERS_BRANCH=2018.0.0.17900

# Skip Java 8 builds on distributions that don't support it
#export BLD_JAVA_8=

# POST_NIGHTLY can be any command to be run at the end of nightly.  See
# nightly(1) for interactions between environment variables and this command.
#POST_NIGHTLY=

# ------------------------------------------------------------------------------
# You are less likely to need to modify parameters below.
# ------------------------------------------------------------------------------

# Maximum number of dmake jobs.  The recommended number is 2 + NCPUS,
```

```
127 # where NCPUS is the number of logical CPUs on your build system.
128 function maxjobs
129 {
130         nameref maxjobs=$1
131         integer ncpu
132         integer -r min_mem_per_job=512 # minimum amount of memory for a job

134         ncpu=$(builtin getconf ; getconf 'NPROCESSORS_ONLN')
135         (( maxjobs=ncpu + 2 ))

137         # Throttle number of parallel jobs launched by dmake to a value which
138         # gurantees that all jobs have enough memory. This was added to avoid
139         # excessive paging/swapping in cases of virtual machine installations
140         # which have lots of CPUs but not enough memory assigned to handle
141         # that many parallel jobs
142         if [[ $(/usr/sbin/prtconf 2>'/dev/null') == ~(E)Memory\ size:\ ([[:digit
143                 integer max_jobs_per_memory # parallel jobs which fit into physi
144                 integer physical_memory # physical memory installed

146                 # The array ".sh.match" contains the contents of capturing
147                 # brackets in the last regex, .sh.match[1] will contain
148                 # the value matched by ([[:digit:]]+), i.e. the amount of
149                 # memory installed
150                 physical_memory="10#${.sh.match[1]}"

152                 ((
153                         max_jobs_per_memory=round(physical_memory/min_mem_per_jo
154                         maxjobs=fmax(2, fmin(maxjobs, max_jobs_per_memory))
155                 ))
156         fi

158         return 0
159 }

161 maxjobs DMAKE_MAX_JOBS # "DMAKE_MAX_JOBS" passed as ksh(1) name reference
162 export DMAKE_MAX_JOBS

164 # path to onbld tool binaries
165 ONBLD_BIN='/opt/onbld/bin'

167 # PARENT_WS is used to determine the parent of this workspace. This is
168 # for the options that deal with the parent workspace (such as where the
169 # proto area will go).
170 export PARENT_WS=''

172 # CLONE_WS is the workspace nightly should do a bringover from.
173 # The bringover, if any, is done as STAFFER.
174 export CLONE_WS='ssh://anonhg@hg.illumos.org/illumos-gate'

176 # Set STAFFER to your own login as gatekeeper or developer
177 # The point is to use group "staff" and avoid referencing the parent
178 # workspace as root.
179 export STAFFER="$LOGNAME"
180 export MAILTO="${MAILTO:-$STAFFER}"

182 # If you wish the mail messages to be From: an arbitrary address, export
183 # MAILFROM.
184 #export MAILFROM="user@example.com"

186 # The project (see project(4)) under which to run this build.  If not
187 # specified, the build is simply run in a new task in the current project.
188 export BUILD_PROJECT=''

190 # You should not need to change the next three lines
191 export ATLOG="$CODEMGR_WS/log"
192 export LOGFILE="$ATLOG/nightly.log"
```



```
193 export MACH="$(uname -p)"

195 #
196 #  The following macro points to the closed binaries.  Once illumos has
197 #  totally freed itself, we can remove this reference.
198 #
199 # Location of encumbered binaries.
200 export ON_CLOSED_BINS="$CODEMGR_WS/closed"

202 # REF_PROTO_LIST - for comparing the list of stuff in your proto area
203 # with. Generally this should be left alone, since you want to see differences
204 # from your parent (the gate).
205 #
206 export REF_PROTO_LIST="$PARENT_WS/usr/src/proto_list_${MACH}"


209 export ROOT="$CODEMGR_WS/proto/root_${MACH}"
210 export SRC="$CODEMGR_WS/usr/src"
211 export MULTI_PROTO="no"

213 #
214 #       build environment variables, including version info for mcs, motd,
215 # motd, uname and boot messages. Mostly you shouldn't change this except
216 # when the release slips (nah) or you move an environment file to a new
217 # release
218 #
219 export VERSION="'git describe --long --all HEAD | cut -d/ -f2-'"

221 #
222 # the RELEASE and RELEASE_DATE variables are set in Makefile.master;
223 # there might be special reasons to override them here, but that
224 # should not be the case in general
225 #
226 # export RELEASE='5.11'
227 # export RELEASE_DATE='October 2007'

229 # proto area in parent for optionally depositing a copy of headers and
230 # libraries corresponding to the protolibs target
231 # not applicable given the NIGHTLY_OPTIONS
232 #
233 export PARENT_ROOT="$PARENT_WS/proto/root_$MACH"
234 export PARENT_TOOLS_ROOT="$PARENT_WS/usr/src/tools/proto/root_$MACH-nd"

236 # Package creation variables.  You probably shouldn't change these,
237 # either.
238 #
239 # PKGARCHIVE determines where the repository will be created.
240 #
241 # PKGPUBLISHER_REDIST controls the publisher setting for the repository.
242 #
243 export PKGARCHIVE="${CODEMGR_WS}/packages/${MACH}/nightly"
244 # export PKGPUBLISHER_REDIST='on-redist'

246 # Package manifest format version.
247 export PKGFMT_OUTPUT='v1'

249 # we want make to do as much as it can, just in case there's more than
250 # one problem.
251 export MAKEFLAGS='k'

253 # Magic variables to prevent the devpro compilers/teamware from checking
254 # for updates or sending mail back to devpro on every use.
255 export SUNW_NO_UPDATE_NOTIFY='1'
256 export UT_NO_USAGE_TRACKING='1'

258 # Build tools - don't change these unless you know what you're doing.  These
```

```
259 # variables allows you to get the compilers and onbld files locally.
260 # Set BUILD_TOOLS to pull everything from one location.
261 # Alternately, you can set ONBLD_TOOLS to where you keep the contents of
262 # SUNWonbld and SPRO_ROOT to where you keep the compilers.  SPRO_VROOT
263 # exists to make it easier to test new versions of the compiler.
264 export BUILD_TOOLS='/opt'
265 #export ONBLD_TOOLS='/opt/onbld'
266 export SPRO_ROOT='/opt/SUNWspro'
267 export SPRO_VROOT="$SPRO_ROOT"

269 # This goes along with lint - it is a series of the form "A [y|n]" which
270 # means "go to directory A and run 'make lint'" Then mail me (y) the
271 # difference in the lint output. 'y' should only be used if the area you're
272 # linting is actually lint clean or you'll get lots of mail.
273 # You shouldn't need to change this though.
274 #export LINTDIRS="$SRC y"

276 # Set this flag to 'n' to disable the use of 'checkpaths'.  The default,
277 # if the 'N' option is not specified, is to run this test.
278 #CHECK_PATHS='y'

280 if [[ "$ENABLE_SMATCH" = "1" ]]; then
281         SMATCHBIN=$CODEMGR_WS/usr/src/tools/proto/root_$MACH-nd/opt/onbld/bin/$M
282         export SHADOW_CCS="$SHADOW_CCS smatch,$SMATCHBIN,smatch"
283 fi
```

```
**********************************************************
    3039 Fri Dec 21 14:59:58 2018
new/usr/src/tools/scripts/checkpaths.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#!/bin/ksh -p
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.

# Quis custodiet ipsos custodies?

if [ -z "$SRC" ]; then
        SRC=$CODEMGR_WS/usr/src
fi

if [ -z "$CODEMGR_WS" -o ! -d "$CODEMGR_WS" -o ! -d "$SRC" ]; then
        echo "$0: must be run from within a workspace."
        exit 1
fi

cd $CODEMGR_WS || exit 1

# Use -b to tell this script to ignore derived (built) objects.
if [ "$1" = "-b" ]; then
        b_flg=y
fi

# Not currently used; available for temporary workarounds.
args="-k NEVER_CHECK"

# We intentionally don't depend on $MACH here, and thus no $ROOT.  If
# a proto area exists, then we use it.  This allows this script to be
# run against gates (which should contain both SPARC and x86 proto
# areas), build workspaces (which should contain just one proto area),
# and unbuilt workspaces (which contain no proto areas).
if [ "$b_flg" = y ]; then
        rootlist=
elif [ $# -gt 0 ]; then
        rootlist=$*
else
        rootlist="$CODEMGR_WS/proto/root_sparc $CODEMGR_WS/proto/root_i386"
```



```
fi

for ROOT in $rootlist
do
        case "$ROOT" in
        *sparc|*sparc-nd)
                arch=sparc
                ;;
        *i386|*i386-nd)
                arch=i386
                ;;
        *)
                echo "$ROOT has unknown architecture." >&2
                exit 1
                ;;
        esac
        if [ -d $ROOT ]; then
                #
                # This is the old-style packaging exception list, from
                # the svr4-specific usr/src/pkgdefs
                #
                [ -f $SRC/pkgdefs/etc/exception_list_$arch ] && \
                        validate_paths '-s/\s*'$arch'$//'  \
                            -e ^usr/include/ike/ -b $ROOT \
                            $args $SRC/pkgdefs/etc/exception_list_$arch
                #
                # These are the new-style packaging exception lists,
                # from the repository-wide exception_lists/ directory.
                #
                e="$CODEMGR_WS/exception_lists/packaging"
                for f in $e; do
                        if [ -f $f ]; then
                                nawk 'NF == 1 || /[     ]\+'$arch'$/ { print; }'
                                    < $f | validate_paths -b $ROOT -n $f
                        fi
                done
        fi
done

# Two entries in the findunref exception_list deal with things created
# by nightly.  Otherwise, this test could be run on an unmodifed (and
# unbuilt) workspace.  We handle this by flagging the one that is
# present only on a built workspace (./*.out) and the one that's
# present only after a run of findunref (./*.ref) with ISUSED, and
# disabling all checks of them.  The assumption is that the entries
# marked with ISUSED are always known to be good, thus the Latin quote
# at the top of the file.
#
# The exception_list is generated from whichever input files are appropriate
# for this workspace, so checking it obviates the need to check the inputs.

if [ -r $SRC/tools/findunref/exception_list ]; then
        validate_paths -k ISUSED -r -e '^\*' $SRC/tools/findunref/exception_list
fi

if [ -f $SRC/tools/opensolaris/license-list ]; then
        sed -e 's/$/.descrip/' < $SRC/tools/opensolaris/license-list | \
                validate_paths -n SRC/tools/opensolaris/license-list
fi

validate_flg -f

exit 0
```

```
**********************************************************
   58088 Fri Dec 21 14:59:58 2018
new/usr/src/tools/scripts/nightly.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
______unchanged_portion_omitted_

 163 #
 164 # Function to do the build, including package generation.
 165 # usage: build LABEL SUFFIX ND MULTIPROTO
 166 # - LABEL is used to tag build output.
 167 # - SUFFIX is used to distinguish files (e.g., DEBUG vs non-DEBUG,
 168 #   open-only vs full tree).
 169 # - ND is "-nd" (non-DEBUG builds) or "" (DEBUG builds).
 170 # - If MULTIPROTO is "yes", it means to name the proto area according to
 171 #   SUFFIX.  Otherwise ("no"), (re)use the standard proto area.
 172 #
 173 function build {
 174         LABEL=$1
 175         SUFFIX=$2
 176         ND=$3
 177         MULTIPROTO=$4
 178         INSTALLOG=install${SUFFIX}-${MACH}
 179         NOISE=noise${SUFFIX}-${MACH}
 180         PKGARCHIVE=${PKGARCHIVE_ORIG}${SUFFIX}

 182         ORIGROOT=$ROOT
 183         [ $MULTIPROTO = no ] || export ROOT=$ROOT$SUFFIX

 185         export ENVLDLIBS1='myldlibs $ROOT'
 186         export ENVCPPFLAGS1='myheaders $ROOT'

 188         this_build_ok=y
 189         #
 190         #       Build OS-Networking source
 191         #
 192         echo "\n==== Building OS-Net source at 'date' ($LABEL) ====\n" \
 193                 >> $LOGFILE

 195         rm -f $SRC/${INSTALLOG}.out
 196         cd $SRC
 197         /bin/time $MAKE -e install 2>&1 | \
 198             tee -a $SRC/${INSTALLOG}.out >> $LOGFILE

 200         echo "\n==== Build errors ($LABEL) ====\n" >> $mail_msg_file
 201         egrep ":" $SRC/${INSTALLOG}.out |
 202             egrep -e "(^${MAKE}:|[      ]error[:        \n])" | \
 203             egrep -v "Ignoring unknown host" | \
 204             egrep -v "cc .* -o error " | \
 205             egrep -v "warning" | tee $TMPDIR/build_errs${SUFFIX} \
 206             >> $mail_msg_file
 207             sed -n "/^Undefined[        ]*first referenced$/,/^ld: fatal:/p" \
 208             < $SRC/${INSTALLOG}.out >> $mail_msg_file
 209         if [[ -s $TMPDIR/build_errs${SUFFIX} ]]; then
 210                 build_ok=n
 211                 this_build_ok=n
 212         fi
 213         grep "bootblock image is .* bytes too big" $SRC/${INSTALLOG}.out \
 214                 >> $mail_msg_file
 215         if [ "$?" = "0" ]; then
 216                 build_ok=n
 217                 this_build_ok=n
 218         fi

 220         echo "\n==== Build warnings ($LABEL) ====\n" >>$mail_msg_file
```



```
 221         egrep -i 'warn:|warning:' $SRC/${INSTALLOG}.out \
 221         egrep -i warning: $SRC/${INSTALLOG}.out \
 222                 | egrep -v '^tic:' \
 223                 | egrep -v "symbol (\'|')timezone' has differing types:" \
 224                 | egrep -v "parameter <PSTAMP> set to" \
 225                 | egrep -v "Ignoring unknown host" \
 226                 | egrep -v "redefining segment flags attribute for" \
 227                 | tee $TMPDIR/build_warnings${SUFFIX} >> $mail_msg_file
 228         if [[ -s $TMPDIR/build_warnings${SUFFIX} ]]; then
 229                 build_ok=n
 230                 this_build_ok=n
 231         fi

 233         echo "\n==== Ended OS-Net source build at 'date' ($LABEL) ====\n" \
 234                 >> $LOGFILE

 236         echo "\n==== Elapsed build time ($LABEL) ====\n" >>$mail_msg_file
 237         tail -3  $SRC/${INSTALLOG}.out >>$mail_msg_file

 239         if [ "$i_FLAG" = "n" ]; then
 240                 rm -f $SRC/${NOISE}.ref
 241                 if [ -f $SRC/${NOISE}.out ]; then
 242                         mv $SRC/${NOISE}.out $SRC/${NOISE}.ref
 243                 fi
 244                 grep : $SRC/${INSTALLOG}.out \
 245                         | egrep -v '^/' \
 246                         | egrep -v '^(Start|Finish|real|user|sys|./bld_awk)' \
 247                         | egrep -v '^tic:' \
 248                         | egrep -v '^mcs' \
 249                         | egrep -v '^LD_LIBRARY_PATH=' \
 250                         | egrep -v 'ar: creating' \
 251                         | egrep -v 'ar: writing' \
 252                         | egrep -v 'conflicts:' \
 253                         | egrep -v ':saved created' \
 254                         | egrep -v '^stty.*c:' \
 255                         | egrep -v '^mfgname.c:' \
 256                         | egrep -v '^uname-i.c:' \
 257                         | egrep -v '^volumes.c:' \
 258                         | egrep -v '^lint library construction:' \
 259                         | egrep -v 'tsort: INFORM:' \
 260                         | egrep -v 'stripalign:' \
 261                         | egrep -v 'chars, width' \
 262                         | egrep -v "symbol (\'|')timezone' has differing types:"
 263                         | egrep -v 'PSTAMP' \
 264                         | egrep -v '^Manifying' \
 265                         | egrep -v 'Ignoring unknown host' \
 266                         | egrep -v 'Processing method:' \
 267                         | egrep -v '^Writing' \
 268                         | egrep -v 'spellin1:' \
 269                         | egrep -v '^adding:' \
 270                         | egrep -v "^echo 'msgid" \
 271                         | egrep -v '^echo ' \
 272                         | egrep -v '\.c:$' \
 273                         | egrep -v '^Adding file:' \
 274                         | egrep -v 'CLASSPATH=' \
 275                         | egrep -v '\/var\/mail\/:saved' \
 276                         | egrep -v -- '-DUTS_VERSION=' \
 277                         | egrep -v '^Running Mkbootstrap' \
 278                         | egrep -v '^Applet length read:' \
 279                         | egrep -v 'bytes written:' \
 280                         | egrep -v '^File:SolarisAuthApplet.bin' \
 281                         | egrep -v -i 'jibversion' \
 282                         | egrep -v '^Output size:' \
 283                         | egrep -v '^Solo size statistics:' \
 284                         | egrep -v '^Using ROM API Version' \
 285                         | egrep -v '^Zero Signature length:' \
```

```
286                         | egrep -v '^Note \(probably harmless\):' \
287                         | egrep -v '::' \
288                         | egrep -v '^\+' \
289                         | egrep -v 'svccfg-native -s svc:/' \
290                         | sort | uniq >$SRC/${NOISE}.out
291                 if [ ! -f $SRC/${NOISE}.ref ]; then
292                         cp $SRC/${NOISE}.out $SRC/${NOISE}.ref
293                 fi
294                 echo "\n==== Build noise differences ($LABEL) ====\n" \
295                         >>$mail_msg_file
296                 diff $SRC/${NOISE}.ref $SRC/${NOISE}.out >>$mail_msg_file
297         fi

299         #
300         #       Re-sign selected binaries using signing server
301         #       (gatekeeper builds only)
302         #
303         if [ -n "$CODESIGN_USER" -a "$this_build_ok" = "y" ]; then
304                 echo "\n==== Signing proto area at 'date' ====\n" >> $LOGFILE
305                 signing_file="${TMPDIR}/signing"
306                 rm -f ${signing_file}
307                 export CODESIGN_USER
308                 signproto $SRC/tools/codesign/creds 2>&1 | \
309                         tee -a ${signing_file} >> $LOGFILE
310                 echo "\n==== Finished signing proto area at 'date' ====\n" \
311                     >> $LOGFILE
312                 echo "\n==== Crypto module signing errors ($LABEL) ====\n" \
313                     >> $mail_msg_file
314                 egrep 'WARNING|ERROR' ${signing_file} >> $mail_msg_file
315                 if (( $? == 0 )) ; then
316                         build_ok=n
317                         this_build_ok=n
318                 fi
319         fi

321         #
322         #       Building Packages
323         #
324         if [ "$p_FLAG" = "y" -a "$this_build_ok" = "y" ]; then
325                 if [ -d $SRC/pkg ]; then
326                         echo "\n==== Creating $LABEL packages at 'date' ====\n"
327                                 >> $LOGFILE
328                         echo "Clearing out $PKGARCHIVE ..." >> $LOGFILE
329                         rm -rf $PKGARCHIVE >> "$LOGFILE" 2>&1
330                         mkdir -p $PKGARCHIVE >> "$LOGFILE" 2>&1

332                         rm -f $SRC/pkg/${INSTALLOG}.out
333                         cd $SRC/pkg
334                         /bin/time $MAKE -e install 2>&1 | \
335                             tee -a $SRC/pkg/${INSTALLOG}.out >> $LOGFILE

337                         echo "\n==== package build errors ($LABEL) ====\n" \
338                                 >> $mail_msg_file

340                         egrep "${MAKE}|ERROR|WARNING" $SRC/pkg/${INSTALLOG}.out
341                                 grep ':' | \
342                                 grep -v PSTAMP | \
343                                 egrep -v "Ignoring unknown host" | \
344                                 tee $TMPDIR/package >> $mail_msg_file
345                         if [[ -s $TMPDIR/package ]]; then
346                                 build_extras_ok=n
347                                 this_build_ok=n
348                         fi
349                 else
350                         #
351                         # Handle it gracefully if -p was set but there so
```



```
352                         # no pkg directory.
353                         #
354                         echo "\n==== No $LABEL packages to build ====\n" \
355                                 >> $LOGFILE
356                 fi
357         else
358                 echo "\n==== Not creating $LABEL packages ====\n" >> $LOGFILE
359         fi

361         ROOT=$ORIGROOT
362 }
```

_____***unchanged_portion_omitted_***

```
**********************************************************
    5181 Fri Dec 21 14:59:58 2018
new/usr/src/tools/smatch/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#
# Copyright (c) 2018, Joyent, Inc.
#

#
# The src/ sub-directory is un-modified copy of
# https://github.com/illumos/smatch/tree/0.5.1-il-1
#
# This Makefile installs just enough for us to be able to run smatch
# locally.
#

PROG = smatch
SPARSE_VERSION = 0.5.1-il-1

include ../Makefile.tools

# We have to build smatch before we can use cw
i386_CC = $(GNUC_ROOT)/bin/gcc
sparc_CC = $(GNUC_ROOT)/bin/gcc

CFLAGS = -O -D__sun -Wall -Wno-unknown-pragmas -std=gnu99 -nodefaultlibs

SMATCHDATADIR = $(ROOTONBLDSHARE)/smatch

CFLAGS += -DSMATCHDATADIR='"$(SMATCHDATADIR)"'
CFLAGS += -DGCC_BASE='"/no/such/dir"'
CFLAGS += -DMULTIARCH_TRIPLET=NULL

LDLIBS += -lsqlite3 -lcrypto -lm -lgcc -lc
LDFLAGS = $(MAPFILE.NES:%=-Wl,-M%)
LDFLAGS += -L$(NATIVE_ADJUNCT)/lib -R$(NATIVE_ADJUNCT)/lib

CPPFLAGS += -nostdinc
CPPFLAGS += -Isrc/
CPPFLAGS += -I$(NATIVE_ADJUNCT)/include

# no install.bin
INS.file = $(RM) $@; $(CP) $< $(@D); $(CHMOD) $(FILEMODE) $@
INS.dir = mkdir -p $@; $(CHMOD) $(FILEMODE) $@

SMATCH_CHECK_OBJS:sh=ls src/check_*.c | sed -e 's+\.c+.o+;s+src/++;'

OBJS = smatch.o $(SMATCH_CHECK_OBJS)

OBJS += smatch_flow.o smatch_conditions.o smatch_slist.o smatch_states.o \
        smatch_helper.o smatch_type.o smatch_hooks.o smatch_function_hooks.o \
        smatch_modification_hooks.o smatch_extra.o smatch_estate.o smatch_math.o
        smatch_sval.o smatch_ranges.o smatch_implied.o smatch_ignore.o smatch_pr
        smatch_var_sym.o smatch_tracker.o smatch_files.o smatch_expression_stack
        smatch_equiv.o smatch_buf_size.o smatch_strlen.o smatch_capped.o smatch_
```



```
        smatch_expressions.o smatch_returns.o smatch_parse_call_math.o \
        smatch_param_limit.o smatch_param_filter.o \
        smatch_param_set.o smatch_comparison.o smatch_param_compare_limit.o smat
        smatch_function_ptrs.o smatch_annotate.o smatch_string_list.o \
        smatch_param_cleared.o smatch_start_states.o \
        smatch_recurse.o smatch_data_source.o smatch_type_val.o \
        smatch_common_functions.o smatch_struct_assignment.o \
        smatch_unknown_value.o smatch_stored_conditions.o avl.o \
        smatch_function_info.o smatch_links.o smatch_auto_copy.o \
        smatch_type_links.o smatch_untracked_param.o smatch_impossible.o \
        smatch_strings.o smatch_param_used.o smatch_container_of.o smatch_addres
        smatch_buf_comparison.o smatch_real_absolute.o smatch_scope.o \
        smatch_imaginary_absolute.o smatch_parameter_names.o \
        smatch_return_to_param.o smatch_passes_array_size.o \
        smatch_constraints.o smatch_constraints_required.o \
        smatch_fn_arg_link.o smatch_about_fn_ptr_arg.o smatch_mtag.o \
        smatch_mtag_map.o smatch_mtag_data.o \
        smatch_param_to_mtag_data.o smatch_mem_tracker.o smatch_array_values.o \
        smatch_nul_terminator.o smatch_assigned_expr.o smatch_kernel_user_data.o
        smatch_statement_count.o

OBJS += target.o parse.o tokenize.o pre-process.o symbol.o lib.o scope.o \
        expression.o show-parse.o evaluate.o expand.o inline.o linearize.o \
        char.o sort.o allocate.o compat-linux.o ptrlist.o \
        builtin.o \
        stats.o \
        flow.o cse.o simplify.o memops.o liveness.o storage.o unssa.o \
        dissect.o \
        macro_table.o token_store.o hashtable.o

SMATCH_DATA = \
        illumos_kernel.no_return_funcs \
        illumos_kernel.skipped_functions \
        illumos_user.no_return_funcs \
        illumos_user.skipped_functions

SMATCH_DB_DATA = \
        return_states.schema \
        call_implies.schema \
        type_value.schema \
        param_map.schema \
        function_type_size.schema \
        parameter_name.schema \
        fn_ptr_data_link.schema \
        constraints.schema \
        mtag_about.schema \
        type_info.schema \
        function_type_info.schema \
        caller_info.schema \
        function_type_value.schema \
        return_implies.schema \
        type_size.schema \
        constraints_required.schema \
        fn_data_link.schema \
        mtag_alias.schema \
        common_caller_info.schema \
        data_info.schema \
        function_type.schema \
        db.schema \
        mtag_data.schema \
        function_ptr.schema \
        sink_info.schema \
        local_values.schema \
        mtag_map.schema

ROOTONBLDDATAFILES = $(SMATCH_DATA:%=$(SMATCHDATADIR)/smatch_data/%)
```

```
127 ROOTONBLDDATAFILES += $(SMATCH_DB_DATA:%=$(SMATCHDATADIR)/smatch_data/db/%)

129 BUILT_HEADERS = src/version.h src/check_list_local.h

131 .KEEP_STATE:

133 all: $(PROG)

135 install: all .WAIT $(ROOTONBLDMACHPROG) $(ROOTONBLDDATAFILES)

137 clean:
138         rm -f $(OBJS) $(BUILT_HEADERS)

140 $(ROOTONBLDDATAFILES): $(SMATCHDATADIR)/smatch_data/db

142 $(SMATCHDATADIR)/smatch_data/%: src/smatch_data/%
143         $(INS.file)

145 $(SMATCHDATADIR)/smatch_data/db:
146         $(INS.dir)

148 $(SMATCHDATADIR)/smatch_data:
149         $(INS.dir)

151 $(PROG): $(OBJS)
152         $(LINK.c) $(OBJS) -o $@ $(LDLIBS)
153         $(POST_PROCESS)

155 %.o: src/%.c $(BUILT_HEADERS)
156         $(COMPILE.c) -o $@ $<

158 %.o: src/cwchash/%.c
159         $(COMPILE.c) -o $@ $<

161 src/check_list_local.h:
162         touch src/check_list_local.h

164 src/version.h:
165         echo '#define SPARSE_VERSION "$(SPARSE_VERSION)"' > src/version.h

167 include ../Makefile.targ
```

**********
4323 Fri Dec 21 14:59:58 2018
new/usr/src/tools/smatch/src/Documentation/data-structures.txt
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
- A slightly edited irc discussion with Josh Triplett.
- Describes most data structures used in sparse.

As far as the parsing structures go...
The C parser exists in two main files: parse.c, which parses statements, and exp
parse.h contains the definition of struct statement, which represents a C statem
That includes only those things which can't appear as an expression, which prima
expression.h contains the definition of struct expression, which represents a C
A series of statements forms a compound statement (STMT_COMPOUND).
That appears as another struct statement which has a statement_list member.
A function body consists of a compound statement.
When you look at a loop body, if or else body, or case body, you'll notice that
Also note that all loops get turned into a single "iterator" statement.
for, while, and do-while.
A symbol, then, represents a name in a C file.  A symbol might represent a varia
See symbol.h.
"struct symbol" represents one symbol.
As with the various other structures, it has some common data and a union of sub
Most of the interesting bits come in the NS_SYMBOL case.
Among other things, it has a struct statement for the body of a function (if any
Together, struct symbol, struct statement, and struct expression represent most
So, that represents most of the "front-end" of Sparse: parsing C and generating
That much occurs in pretty much any program using the Sparse frontend.
The backend varies among programs.
For instance, the c2xml backend goes that far, then outputs XML.
The sparse static analysis backend has a few steps: it generates linearized byte
Several other backends run that linearized bytecode stage.
The linearized bytecode itself has a set of nested structures.
linearize.h defines all of them.
At the top level, it has struct entrypoint.
That represents an entrypoint to the code, which would normally mean a function.
An entrypoint has a list of basic blocks.
struct basic_block.
A basic block represents a series of instructions with no branches.
Straight-line code.
A branch only occurs at the end of a basic block, and branches can only target t
Typically, a conditional will consist of a basic block leading up to the branch,
Either the true or the false case may not exist.
A loop will normally have a basic block for the loop body, which can branch to t
So basic blocks represent a node in the control flow graph.
The edges in that graph lead from one basic block to a basic block which can fol
Each basic block has a series of instructions, "struct instruction".
"enum opcode" lists all the instructions.
Fairly high-level instruction set, corresponding directly to bits of C.
So you have an entrypoint, which has a graph of basic blocks, each of which has
An entrypoint also has a pointer to the first instruction.
One last bit of trickiness: struct pseudo.
Have you ever heard of "static single assignment" or SSA form?
struct pseudo represents one of those single-assignment variables.
Each one has a pointer to the symbol it represents (which may have many pseudos
Each one also has a pointer to the instruction that defines it.
That covers most of the major data structures in Sparse.
Now, given all that, some of the top-level stuff in sparse.c may make more sense
For instance, the context checking works in terms of basic blocks.
Hopefully some of that helped you understand Sparse better.
```

**********
*2053 Fri Dec 21 14:59:59 2018*
*new/usr/src/tools/smatch/src/Documentation/project-ideas.md*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
Why hacking on sparse
=====================

1. sparse is small.
   The full project compiles in less than 10 seconds on old and not performing l
2. sparse is fast.
   Typically, sparse can check a C file 1/10 of time it takes for gcc to generat
3. sparse can digest the full kernel source files.
   With sparse-llvm, sparse uses llvm as back end to emit real machine code.

New developer hacking on sparse
===============================


* All sparse warning messages should include the option how
   to disable it.
       e.g. "pre-process.c:20*:28: warning: Variable length array is used."
       should be something like
        "pre-process.c:20*:28: warning: Variable length array is
used. (-Wno-vla)"
* extend test-inspect to inspect more AST fields.
* extend test-inspect to inspect instructions.
* adding architecture handling in sparse similar to cgcc
* parallel processing of test-suite
* Howto: fix the kernel rcu related checker warnings
* option to disable AST level inline.
* debug: debug version of sparse do all the verification double check
* test suite: verify and compare IR (suggested by Dibyendu Majumdar)
* checker error output database

For experienced developers
==========================

* merge C type on incremental declare of C type and function prototype.
* move attribute out of ctype to allow easier to add new attribute.
* serialize, general object walking driven by data structures.
* serialize, write sparse byte code into file
* serialize, load sparse byte code from file.
* symbol index/linker, know which symbol in which byte code file.
* inline function in instruction level
* cross function checking
* debug: optimization step by step log
* debug: fancy animation of CFG
* phi node location (Luc has patch)
* revisit crazy programmer warning, invalid SSA form.
* ptrlist, looping while modify inside the loop.
* dead code elimination using ssa
* constant propagation using ssa.
* x86/arm back end instruction set define
* register allocation.
* emit x86/arm machine level code
```

```
**********************************************************
    2504 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/Documentation/smatch.txt
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
Smatch

1.  Building Smatch
2.  Using Smatch
3.  Smatch vs Sparse


Section 1:  Building Smatch
---------------------------

Smatch requires sqlite3.  It requires the binaries and the C, Perl and
Python libraries for sqlite3.

apt-get install sqlite3 libsqlite3-dev libdbd-sqlite3-perl

Smatch is easy to build.  Just type 'make'.  There isn't an install process
right now so just run it from the build directory.


Section 2:  Using Smatch
------------------------

Smatch can be used with a cross function database. It's not mandatory to
build the database but it's a useful thing to do.  Building the database
for the kernel takes 2-3 hours on my computer.  For the kernel you build
the database with:

        cd ~/path/to/kernel_dir
        ~/path/to/smatch_dir/smatch_scripts/build_kernel_data.sh

For projects other than the kernel you run Smatch with the options
"--call-tree --info --param-mapper --spammy" and finish building the
database by running the script:

        ~/progs/smatch/devel/smatch_data/db/create_db.sh

Each time you rebuild the cross function database it becomes more accurate. I
normally rebuild the database every morning.

If you are running Smatch over the whole kernel you can use the following
command:

        ~/progs/smatch/devel/smatch_scripts/test_kernel.sh

The test_kernel.sh script will create a .c.smatch file for every file it tests
and a combined smatch_warns.txt file with all the warnings.

If you are running Smatch just over one kernel file:

        ~/progs/smatch/devel/smatch_scripts/kchecker drivers/whatever/file.c

You can also build a directory like this:

        ~/progs/smatch/devel/smatch_scripts/kchecker drivers/whatever/

The kchecker script prints its warnings to stdout.

If you are building something else (which is not the Linux kernel) then use
something like:
```



```
        make CHECK="~/progs/smatch/devel/smatch --full-path" \
                CC=~/progs/smatch/devel/smatch/cgcc | tee smatch_warns.txt

The makefile has to let people set the CC with an environment variable for that
to work, of course.


Section 3:  Smatch vs Sparse
----------------------------

Smatch uses Sparse as a C parser.  I have made a few hacks to Sparse so I
have to distribute the two together.  Sparse is released under the MIT license
and Smatch is GPLv2+.  If you make changes to Sparse please send those to the
Sparse mailing list linux-sparse@vger.kernel.org and I will pick them up from
there.  Partly I do that for licensing reasons because I don't want to pull GPL
changes into the Sparse code I re-distribute.
```

```
**********************************************************
    3019 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/Documentation/sparse-README.txt
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************

  sparse (spˆ/rs), adj,., spars-er, spars-est.
        1. thinly scattered or distributed; "a sparse population"
        2. thin; not thick or dense: "sparse hair"
        3. scanty; meager.
        4. semantic parse
        [ from Latin: spars(us) scattered, past participle of
          spargere 'to sparge' ]

        Antonym: abundant

Sparse is a semantic parser of source files: it's neither a compiler
(although it could be used as a front-end for one) nor is it a
preprocessor (although it contains as a part of it a preprocessing
phase).

It is meant to be a small - and simple - library.  Scanty and meager,
and partly because of that easy to use.  It has one mission in life:
create a semantic parse tree for some arbitrary user for further
analysis.  It's not a tokenizer, nor is it some generic context-free
parser.  In fact, context (semantics) is what it's all about - figuring
out not just what the grouping of tokens are, but what the _types_ are
that the grouping implies.

And no, it doesn't use lex and yacc (or flex and bison).  In my personal
opinion, the result of using lex/yacc tends to end up just having to
fight the assumptions the tools make.

The parsing is done in five phases:

 - full-file tokenization
 - pre-processing (which can cause another tokenization phase of another
   file)
 - semantic parsing.
 - lazy type evaluation
 - inline function expansion and tree simplification

Note the "full file" part. Partly for efficiency, but mostly for ease of
use, there are no "partial results". The library completely parses one
whole source file, and builds up the _complete_ parse tree in memory.

Also note the "lazy" in the type evaluation.  The semantic parsing
itself will know which symbols are typedefines (required for parsing C
correctly), but it will not have calculated what the details of the
different types are.  That will be done only on demand, as the back-end
requires the information.

This means that a user of the library will literally just need to do

  struct string_list *filelist = NULL;
  char *file;

  action(sparse_initialize(argc, argv, filelist));

  FOR_EACH_PTR_NOTAG(filelist, file) {
    action(sparse(file));
  } END_FOR_EACH_PTR_NOTAG(file);

and he is now done - having a full C parse of the file he opened.  The
library doesn't need any more setup, and once done does not impose any
```



```
more requirements.  The user is free to do whatever he wants with the
parse tree that got built up, and needs not worry about the library ever
again.  There is no extra state, there are no parser callbacks, there is
only the parse tree that is described by the header files. The action
funtion takes a pointer to a symbol_list and does whatever it likes with it.

The library also contains (as an example user) a few clients that do the
preprocessing, parsing and type evaluation and just print out the
results.  These clients were done to verify and debug the library, and
also as trivial examples of what you can do with the parse tree once it
is formed, so that users can see how the tree is organized.
```

```
**********************************************************
    1827 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/Documentation/sparse.txt
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
Sparse
~~~~~~

__nocast vs __bitwise:

__nocast warns about explicit or implicit casting to different types.

HOWEVER, it doesn't consider two 32-bit integers to be different
types, so a __nocast 'int' type may be returned as a regular 'int'
type and then the __nocast is lost.

So "__nocast" on integer types is usually not that powerful. It just
gets lost too easily. It's more useful for things like pointers. It
also doesn't warn about the mixing: you can add integers to __nocast
integer types, and it's not really considered anything wrong.

__bitwise ends up being a "stronger integer separation". That one
doesn't allow you to mix with non-bitwise integers, so now it's much
harder to lose the type by mistake.

So the basic rule is:

 - "__nocast" on its own tends to be more useful for *big* integers
that still need to act like integers, but you want to make it much
less likely that they get truncated by mistake. So a 64-bit integer
that you don't want to mistakenly/silently be returned as "int", for
example. But they mix well with random integer types, so you can add
to them etc without using anything special. However, that mixing also
means that the __nocast really gets lost fairly easily.

 - "__bitwise" is for *unique types* that cannot be mixed with other
types, and that you'd never want to just use as a random integer (the
integer 0 is special, though, and gets silently accepted iirc - it's
kind of like "NULL" for pointers). So "gfp_t" or the "safe endianness"
types would be __bitwise: you can only operate on them by doing
specific operations that know about *that* particular type.

Generally, you want __bitwise if you are looking for type safety.
"__nocast" really is pretty weak.

Reference:

* Linus' e-mail about __nocast vs __bitwise:

  http://marc.info/?l=linux-mm&m=133245421127324&w=2
```

**********
831 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/Documentation/submitting-patches.md
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
Submitting patches: the sparse version
======================================

Sparse uses a patch submit process similar to the Linux Kernel
[Submitting Patches](https://www.kernel.org/doc/html/v4.12/process/submitting-pa

This document mostly focuses on the parts that might be different from the Linux
Kernel submitting process.

1. Git clone a sparse repository:

        git clone git://git.kernel.org/pub/scm/devel/sparse/sparse.git

2. [Coding Style](https://www.kernel.org/doc/html/v4.12/process/coding-style.htm

3. Sign off the patch.

   The usage of the Signed-off-by tag is the same as [Linux Kernel Sign your wor

   Notice that sparse uses the MIT License.
```

```
**********************************************************
    3852 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/Documentation/test-suite
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  3         Sparse test suite
  4         ~~~~~~~~~~~~~~~~~

  6 Sparse has a number of test cases in its validation directory. The test-suite
  7 script aims at making automated checking of these tests possible. It works by
  8 embedding tags in C comments in the test cases.

 10 check-name: (mandatory)
 11         Name of the test.

 13 check-description: (optional)
 14         A description of what the test checks.

 16 check-command: (optional)
 17         There are different kinds of tests. Some can validate the sparse
 18         preprocessor, while others will use sparse, cgcc, or even other backends
 19         of the library. check-command allows you to give a custom command to
 20         run the test-case.
 21         The '$file' string is special. It will be expanded to the file name at
 22         run time.
 23         It defaults to "sparse $file".

 25 check-exit-value: (optional)
 26         The expected exit value of check-command. It defaults to 0.

 28 check-timeout: (optional)
 29         The maximum expected duration of check-command, in seconds.
 30         It defaults to 1.

 32 check-output-start / check-output-end (optional)
 33         The expected output (stdout and stderr) of check-command lies between
 34         those two tags. It defaults to no output.

 36 check-output-ignore / check-error-ignore (optional)
 37         Don't check the expected output (stdout or stderr) of check-command
 38         (useful when this output is not comparable or if you're only interested
 39         in the exit value).
 40         By default this check is done.

 42 check-known-to-fail (optional)
 43         Mark the test as being known to fail.

 45 check-output-contains: <pattern> (optional)
 46         Check that the output (stdout) contains the given pattern.
 47         Several such tags can be given, in which case the output
 48         must contains all the patterns.

 50 check-output-excludes: <pattern> (optional)
 51         Similar than the above one, but with opposite logic.
 52         Check that the output (stdout) doesn't contain the given pattern.
 53         Several such tags can be given, in which case the output
 54         must contains none of the patterns.

 56 check-output-pattern-<nbr>-times: <pattern> (optional)
 57         Similar to the contains/excludes above, but with full control
 58         of the number of times the pattern should occur in the output.

 60         Using test-suite
```



```
 61         ~~~~~~~~~~~~~~~~

 63 The test-suite script is called through the check target of the Makefile. It
 64 will try to check every test case it finds (find validation -name '*.c').

 66 It can be called to check a single test with:
 67 $ cd validation
 68 $ ./test-suite single preprocessor/preprocessor1.c
 69      TEST     Preprocessor #1 (preprocessor/preprocessor1.c)
 70 preprocessor/preprocessor1.c passed !


 73         Writing a test
 74         ~~~~~~~~~~~~~~

 76 test-suite comes with a format command to make a test easier to write:

 78         test-suite format file [name [cmd]]

 80 name:
 81         check-name value. If no name is provided, it defaults to the file name.
 82 cmd:
 83         check-command value. If no cmd is provided, it defaults to
 84         "sparse $file".

 86 The output of the test-suite format command can be redirected into the
 87 test case to create a test-suite formatted file.

 89 $ ./test-suite format bad-assignment.c Assignment >> bad-assignment.c
 90 $ cat !$
 91 cat bad-assignment.c
 92 /*
 93  * check-name: bad assignment
 94  *
 95  * check-command: sparse $file
 96  * check-exit-value: 1
 97  *
 98  * check-output-start
 99 bad-assignment.c:3:6: error: Expected ; at end of statement
100 bad-assignment.c:3:6: error: got \
101  * check-output-end
102  */

104 You can define the check-command you want to use for the test. $file will be
105 extended to the file name at run time.

107 $ ./test-suite format validation/preprocessor2.c "Preprocessor #2" \
108                  "sparse -E \$file" >> validation/preprocessor2.c
109 $ cat !$
110 cat validation/preprocessor2.c
111 /*
112  * This one we happen to get right.
113  *
114  * It should result in a simple
115  *
116  *      a + b
117  *
118  * for a proper preprocessor.
119  */
120 #define TWO a, b

122 #define UNARY(x) BINARY(x)
123 #define BINARY(x, y) x + y

125 UNARY(TWO)
126 /*
```

```
 * check-name: Preprocessor #2
 *
 * check-command: sparse -E $file
 * check-exit-value: 0
 *
 * check-output-start

a + b
 * check-output-end
 */
```

```
**********************************************************
    3391 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/FAQ
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
        FAQ - Why sparse?

Q.  Why not just use gcc?

A.  Gcc is big, complex, and the gcc maintainers are not interested in
    other uses of the gcc front-end.  In fact, gcc has explicitly
    resisted splitting up the front and back ends and having some common
    intermediate language because of religious license issues - you can
    have multiple front ends and back ends, but they all have to be part
    of gcc and licensed under the GPL.

    This all (in my opinion) makes gcc development harder than it should
    be, and makes the end result very ungainly.  With "sparse", the
    front-end is very explicitly separated into its own independent
    project, and is totally independent from the users.  I don't want to
    know what you do in the back-end, because I don't think I _should_
    know or care.


Q.  Why not GPL?

A.  See the previous question: I personally think that the front end
    must be a totally separate project from the back end: any other
    approach just leads to insanity.  However, at the same time clearly
    we cannot write intermediate files etc crud (since then the back end
    would have to re-parse the whole thing and would have to have its
    own front end and just do a lot of things that do not make any sense
    from a technical standpoint).

    I like the GPL, but as rms says, "Linus is just an engineer". I
    refuse to use a license if that license causes bad engineering
    decisions.  I want the front-end to be considered a separate
    project, yet the GPL considers the required linking to make the
    combined thing a derived work. Which is against the whole point
    of 'sparse'.

    I'm not interested in code generation. I'm not interested in what
    other people do with their back-ends.  I _am_ interested in making a
    good front-end, and "good" means that people find it usable. And
    they shouldn't be scared away by politics or licenses. If they want
    to make their back-end be BSD/MIT licensed, that's great. And if
    they want to have a proprietary back-end, that's ok by me too. It's
    their loss, not mine.


Q.  Does it really parse C?

A.  Yeah, well...  It parses a fairly complete subset of "extended C" as
    defined by gcc.  HOWEVER, since I don't believe in K&R syntax for
    function declarations or in giving automatic integer types, it
    doesn't do that.  If you don't give types to your variables, they
    won't have any types, and you can't use them.

    Similarly, it will be very unhappy about undeclared functions,
    rather than just assuming they have type "int".

    Note that a large rationale for me doing this project is for type
    following, which to some degree explains why the thing is type-anal
    and refuses to touch the old-style pre-ANSI non-typed (or weakly
    typed) constructs. Maybe somebody else who is working on projects
```



```
    where pre-ANSI C makes sense might be more inclined to care about
    ancient C.  It's open source, after all. Go wild.


Q.  What other sparse resources are available?

A.  Wiki: http://sparse.wiki.kernel.org/index.php/Main_Page

    Mailing list: linux-sparse@vger.kernel.org
    See http://vger.kernel.org/vger-lists.html#linux-sparse for subscription
    instructions and links to archives

    Git repo: git://git.kernel.org/pub/scm/devel/sparse/sparse.git
    gitweb: http://git.kernel.org/?p=devel/sparse/sparse.git
```

**********
*18092 Fri Dec 21 14:59:59 2018*
*new/usr/src/tools/smatch/src/GPL-2*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

# GNU GENERAL PUBLIC LICENSE
## Version 2, June 1991



## Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

## GNU GENERAL PUBLIC LICENSE
## TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION



0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

    c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or

collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

  3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer to distribute corresponding source code.  (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it.  For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable.  However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License.  Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

  5. You are not required to accept this License, since you have not signed it.  However, nothing else grants you permission to modify or distribute the Program or its derivative works.  These actions are prohibited by law if you do not accept this License.  Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions.  You may not impose any further



restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

  7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License.  If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all.  For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices.  Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded.  In such case, this License incorporates the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time.  Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number.  If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation.  If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

  10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission.  For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this.  Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than 'show w' and 'show c'; they could even be
mouse-clicks or menu items--whatever suits your program.



You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  'Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.

```
**********************************************************
    3160 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/LICENSE
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
Smatch is released under the GPLv2+ license however Smatch includes the Sparse
source as well which is licensed under the MIT license and cwchash which is
licensed under the three clause BSD license and avl.[ch] which are MIT licensed.

The line between which files are a part of Sparse and which are a part of Smatch
is normally pretty clear.  The one thing I will clarify is that there are some
modifications to Sparse files in the Smatch repository which have not been
pushed to the Sparse repository.  These are released under the MIT license.  The
token_store.c file is a Sparse feature and thus released under the MIT license
despite that it has not been pushed to the Sparse repository.

I will not accept additional Sparse features except under the MIT license.

------  Smatch

License: GPLv2+

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.

See GPL-2, or <http://www.gnu.org/copyleft/gpl.txt> for the terms of the latest
version of the GNU General Public License.

------  Sparse

The 'sparse' C parser front-end library is copyrighted by Transmeta Corp
and other authors and licensed under the "MIT License" as
obtained from www.opensource.org (and included here-in for easy
reference).

[ This copy of the license is the flat-text version of original,
  available in its full glory at

         http://opensource.org/licenses/MIT

  please refer to there for the authoritative and slightly more
  pretty-printed version ]

------

                        The MIT License (MIT)

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:
```



```
The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

```
**********************************************************
   11933 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
VERSION=0.5.1

# Generating file version.h if current version has changed
SPARSE_VERSION:=$(shell git describe 2>/dev/null || echo '$(VERSION)')
VERSION_H := $(shell cat version.h 2>/dev/null)
ifneq ($(lastword $(VERSION_H)),"$(SPARSE_VERSION)")
$(info $(shell echo '     GEN      'version.h))
$(shell echo '#define SPARSE_VERSION "$(SPARSE_VERSION)"' > version.h)
endif

OS = linux

ifeq ($(CC),"")
CC = gcc
endif

CFLAGS += -O2 -finline-functions -fno-strict-aliasing -g
CFLAGS += -Wall -Wwrite-strings -Wno-switch
LDFLAGS += -g -lm -lsqlite3 -lssl -lcrypto
LD = gcc
AR = ar
PKG_CONFIG = pkg-config
COMMON_CFLAGS = -O2 -finline-functions -fno-strict-aliasing -g
COMMON_CFLAGS += -Wall -Wwrite-strings

ALL_CFLAGS = $(COMMON_CFLAGS) $(PKG_CFLAGS) $(CFLAGS)
#
# For debugging, put this in local.mk:
#
#     CFLAGS += -O0 -DDEBUG -g3 -gdwarf-2
#

HAVE_LIBXML:=$(shell $(PKG_CONFIG) --exists libxml-2.0 2>/dev/null && echo 'yes')
HAVE_GCC_DEP:=$(shell touch .gcc-test.c &&                              \
                $(CC) -c -Wp,-MD,.gcc-test.d .gcc-test.c 2>/dev/null && \
                echo 'yes'; rm -f .gcc-test.d .gcc-test.o .gcc-test.c)

GTK_VERSION:=3.0
HAVE_GTK:=$(shell $(PKG_CONFIG) --exists gtk+-$(GTK_VERSION) 2>/dev/null && echo
ifneq ($(HAVE_GTK),yes)
        GTK_VERSION:=2.0
        HAVE_GTK:=$(shell $(PKG_CONFIG) --exists gtk+-$(GTK_VERSION) 2>/dev/null
endif

LLVM_CONFIG:=llvm-config
HAVE_LLVM:=$(shell $(LLVM_CONFIG) --version >/dev/null 2>&1 && echo 'yes')

GCC_BASE := $(shell $(CC) --print-file-name=)
COMMON_CFLAGS += -DGCC_BASE=\"$(GCC_BASE)\"

MULTIARCH_TRIPLET := $(shell $(CC) -print-multiarch 2>/dev/null)
COMMON_CFLAGS += -DMULTIARCH_TRIPLET=\"$(MULTIARCH_TRIPLET)\"

ifeq ($(HAVE_GCC_DEP),yes)
COMMON_CFLAGS += -Wp,-MD,$(@D)/.$(@F).d
endif

DESTDIR=
INSTALL_PREFIX ?=$(HOME)
BINDIR=$(INSTALL_PREFIX)/bin
```



```
LIBDIR=$(INSTALL_PREFIX)/lib
MANDIR=$(INSTALL_PREFIX)/share/man
MAN1DIR=$(MANDIR)/man1
INCLUDEDIR=$(INSTALL_PREFIX)/include
PKGCONFIGDIR=$(LIBDIR)/pkgconfig
SMATCHDATADIR=$(INSTALL_PREFIX)/share/smatch

SMATCH_FILES=smatch_flow.o smatch_conditions.o smatch_slist.o smatch_states.o \
        smatch_helper.o smatch_type.o smatch_hooks.o smatch_function_hooks.o \
        smatch_modification_hooks.o smatch_extra.o smatch_estate.o smatch_math.o
        smatch_sval.o smatch_ranges.o smatch_implied.o smatch_ignore.o smatch_pr
        smatch_var_sym.o smatch_tracker.o smatch_files.o smatch_expression_stack
        smatch_equiv.o smatch_buf_size.o smatch_strlen.o smatch_capped.o smatch_
        smatch_expressions.o smatch_returns.o smatch_parse_call_math.o \
        smatch_param_limit.o smatch_param_filter.o \
        smatch_param_set.o smatch_comparison.o smatch_param_compare_limit.o smat
        smatch_function_ptrs.o smatch_annotate.o smatch_string_list.o \
        smatch_param_cleared.o smatch_start_states.o \
        smatch_recurse.o smatch_data_source.o smatch_type_val.o \
        smatch_common_functions.o smatch_struct_assignment.o \
        smatch_unknown_value.o smatch_stored_conditions.o avl.o \
        smatch_function_info.o smatch_links.o smatch_auto_copy.o \
        smatch_type_links.o smatch_untracked_param.o smatch_impossible.o \
        smatch_strings.o smatch_param_used.o smatch_container_of.o smatch_addres
        smatch_buf_comparison.o smatch_real_absolute.o smatch_scope.o \
        smatch_imaginary_absolute.o smatch_parameter_names.o \
        smatch_return_to_param.o smatch_passes_array_size.o \
        smatch_constraints.o smatch_constraints_required.o \
        smatch_fn_arg_link.o smatch_about_fn_ptr_arg.o smatch_mtag.o \
        smatch_mtag_map.o smatch_mtag_data.o \
        smatch_param_to_mtag_data.o smatch_mem_tracker.o smatch_array_values.o \
        smatch_nul_terminator.o smatch_assigned_expr.o smatch_kernel_user_data.o
        smatch_statement_count.o

SMATCH_CHECKS=$(shell ls check_*.c | sed -e 's/\.c/.o/')
SMATCH_DATA=smatch_data/kernel.allocation_funcs \
        smatch_data/kernel.frees_argument smatch_data/kernel.puts_argument \
        smatch_data/kernel.dev_queue_xmit smatch_data/kernel.returns_err_ptr \
        smatch_data/kernel.dma_funcs smatch_data/kernel.returns_held_funcs \
        smatch_data/kernel.no_return_funcs

SMATCH_SCRIPTS=smatch_scripts/add_gfp_to_allocations.sh \
        smatch_scripts/build_kernel_data.sh \
        smatch_scripts/call_tree.pl smatch_scripts/filter_kernel_deref_check.sh
        smatch_scripts/find_expanded_holes.pl smatch_scripts/find_null_params.sh
        smatch_scripts/follow_params.pl smatch_scripts/gen_allocation_list.sh \
        smatch_scripts/gen_bit_shifters.sh smatch_scripts/gen_dma_funcs.sh \
        smatch_scripts/generisize.pl smatch_scripts/gen_err_ptr_list.sh \
        smatch_scripts/gen_expects_err_ptr.sh smatch_scripts/gen_frees_list.sh \
        smatch_scripts/gen_gfp_flags.sh smatch_scripts/gen_no_return_funcs.sh \
        smatch_scripts/gen_puts_list.sh smatch_scripts/gen_returns_held.sh \
        smatch_scripts/gen_rosenberg_funcs.sh smatch_scripts/gen_sizeof_param.sh
        smatch_scripts/gen_unwind_functions.sh smatch_scripts/kchecker \
        smatch_scripts/kpatch.sh smatch_scripts/new_bugs.sh \
        smatch_scripts/show_errs.sh smatch_scripts/show_ifs.sh \
        smatch_scripts/show_unreachable.sh smatch_scripts/strip_whitespace.pl \
        smatch_scripts/summarize_errs.sh smatch_scripts/test_kernel.sh \
        smatch_scripts/trace_params.pl smatch_scripts/unlocked_paths.pl \
        smatch_scripts/whitespace_only.sh smatch_scripts/wine_checker.sh \

PROGRAMS=test-lexing test-parsing obfuscate compile graph sparse \
         test-linearize example test-unssa test-dissect ctags
INST_PROGRAMS=smatch cgcc

INST_MAN1=sparse.1 cgcc.1
```

```
ifeq ($(HAVE_LIBXML),yes)
PROGRAMS+=c2xml
INST_PROGRAMS+=c2xml
c2xml_EXTRA_OBJS = '$(PKG_CONFIG) --libs libxml-2.0'
LIBXML_CFLAGS := $(shell $(PKG_CONFIG) --cflags libxml-2.0)
else
$(warning Your system does not have libxml, disabling c2xml)
endif

ifeq ($(HAVE_GTK),yes)
GTK_CFLAGS := $(shell $(PKG_CONFIG) --cflags gtk+-$(GTK_VERSION))
GTK_LIBS := $(shell $(PKG_CONFIG) --libs gtk+-$(GTK_VERSION))
PROGRAMS += test-inspect
INST_PROGRAMS += test-inspect
test-inspect_EXTRA_DEPS := ast-model.o ast-view.o ast-inspect.o
test-inspect_OBJS := test-inspect.o $(test-inspect_EXTRA_DEPS)
$(test-inspect_OBJS) $(test-inspect_OBJS:.o=.sc): PKG_CFLAGS += $(GTK_CFLAGS)
test-inspect_EXTRA_OBJS := $(GTK_LIBS)
else
$(warning Your system does not have gtk3/gtk2, disabling test-inspect)
endif

ifeq ($(HAVE_LLVM),yes)
ifeq ($(shell uname -m | grep -q '\(i386\|x86\)' && echo ok),ok)
LLVM_VERSION:=$(shell $(LLVM_CONFIG) --version)
ifeq ($(shell expr "$(LLVM_VERSION)" : '[3-9]\.'),2)
LLVM_PROGS := sparse-llvm
$(LLVM_PROGS): LD := g++
LLVM_LDFLAGS := $(shell $(LLVM_CONFIG) --ldflags)
LLVM_CFLAGS := $(shell $(LLVM_CONFIG) --cflags | sed -e "s/-DNDEBUG//g" | sed -e
LLVM_LIBS := $(shell $(LLVM_CONFIG) --libs)
LLVM_LIBS += $(shell $(LLVM_CONFIG) --system-libs 2>/dev/null)
PROGRAMS += $(LLVM_PROGS)
INST_PROGRAMS += sparse-llvm sparsec
sparse-llvm.o sparse-llvm.sc: PKG_CFLAGS += $(LLVM_CFLAGS)
sparse-llvm_EXTRA_OBJS := $(LLVM_LIBS) $(LLVM_LDFLAGS)
else
$(warning LLVM 3.0 or later required. Your system has version $(LLVM_VERSION) in
endif
else
$(warning sparse-llvm disabled on $(shell uname -m))
endif
else
$(warning Your system does not have llvm, disabling sparse-llvm)
endif

LIB_H=    token.h parse.h lib.h symbol.h scope.h expression.h target.h \
          linearize.h bitmap.h ident-list.h compat.h flow.h allocate.h \
          storage.h ptrlist.h dissect.h

LIB_OBJS= target.o parse.o tokenize.o pre-process.o symbol.o lib.o scope.o \
          expression.o show-parse.o evaluate.o expand.o inline.o linearize.o \
          char.o sort.o allocate.o compat-$(OS).o ptrlist.o \
          builtin.o \
          stats.o \
          flow.o cse.o simplify.o memops.o liveness.o storage.o unssa.o \
          dissect.o \
          macro_table.o token_store.o cwchash/hashtable.o

LIB_FILE= libsparse.a
SLIB_FILE= libsparse.so

# If you add $(SLIB_FILE) to this, you also need to add -fpic to BASIC_CFLAGS ab
# Doing so incurs a noticeable performance hit, and Sparse does not have a
# stable shared library interface, so this does not occur by default.  If you
# really want a shared library, you may want to build Sparse twice: once
```



```
# without -fpic to get all the Sparse tools, and again with -fpic to get the
# shared library.
LIBS=$(LIB_FILE)

#
# Pretty print
#
V             = @
Q             = $(V:1=)
QUIET_CC      = $(Q:@=@echo    '     CC       '$@;)
QUIET_CHECK   = $(Q:@=@echo    '     CHECK    '$<;)
QUIET_AR      = $(Q:@=@echo    '     AR       '$@;)
QUIET_GEN     = $(Q:@=@echo    '     GEN      '$@;)
QUIET_LINK    = $(Q:@=@echo    '     LINK     '$@;)
# We rely on the -v switch of install to print 'file -> $install_dir/file'
QUIET_INST_SH = $(Q:@=@echo -n  '     INSTALL  ';)
QUIET_INST    = $(Q:@=@echo -n '     INSTALL  ';)

define INSTALL_EXEC
        $(QUIET_INST)install -v $1 $(DESTDIR)$2/$1 || exit 1;

endef

define INSTALL_FILE
        $(QUIET_INST)install -v -m 644 $1 $(DESTDIR)$2/$1 || exit 1;

endef

SED_PC_CMD = 's|@version@|$(VERSION)|g;         \
              s|@prefix@|$(INSTALL_PREFIX)|g;          \
              s|@libdir@|$(LIBDIR)|g;           \
              s|@includedir@|$(INCLUDEDIR)|g'



# Allow users to override build settings without dirtying their trees
-include local.mk


all: $(PROGRAMS) sparse.pc smatch

all-installable: $(INST_PROGRAMS) $(LIBS) $(LIB_H) sparse.pc

install: all-installable
        $(Q)install -d $(DESTDIR)$(BINDIR)
        $(Q)install -d $(DESTDIR)$(LIBDIR)
        $(Q)install -d $(DESTDIR)$(MAN1DIR)
        $(Q)install -d $(DESTDIR)$(INCLUDEDIR)/sparse
        $(Q)install -d $(DESTDIR)$(PKGCONFIGDIR)
        $(Q)install -d $(DESTDIR)$(SMATCHDATADIR)/smatch_data
        $(Q)install -d $(DESTDIR)$(SMATCHDATADIR)/smatch_scripts
        $(foreach f,$(INST_PROGRAMS),$(call INSTALL_EXEC,$f,$(BINDIR)))
        $(foreach f,$(INST_MAN1),$(call INSTALL_FILE,$f,$(MAN1DIR)))
        $(foreach f,$(LIBS),$(call INSTALL_FILE,$f,$(LIBDIR)))
        $(foreach f,$(LIB_H),$(call INSTALL_FILE,$f,$(INCLUDEDIR)/sparse))
        $(call INSTALL_FILE,sparse.pc,$(PKGCONFIGDIR))
        $(foreach f,$(SMATCH_DATA),$(call INSTALL_FILE,$f,$(SMATCHDATADIR)))
        $(foreach f,$(SMATCH_SCRIPTS),$(call INSTALL_EXEC,$f,$(SMATCHDATADIR)))

sparse.pc: sparse.pc.in
        $(QUIET_GEN)sed $(SED_PC_CMD) sparse.pc.in > sparse.pc


compile_EXTRA_DEPS = compile-i386.o

$(foreach p,$(PROGRAMS),$(eval $(p): $($(p)_EXTRA_DEPS) $(LIBS)))
```

```
259 $(PROGRAMS): % : %.o
260         $(QUIET_LINK)$(LD) -o $@ $^ $($@_EXTRA_OBJS) $(LDFLAGS)

262 smatch: smatch.o $(SMATCH_FILES) $(SMATCH_CHECKS) $(LIBS)
263         $(QUIET_LINK)$(LD) -o $@ $< $(SMATCH_FILES) $(SMATCH_CHECKS) $(LIBS) $(L

265 $(LIB_FILE): $(LIB_OBJS)
266         $(QUIET_AR)$(AR) rcs $@ $(LIB_OBJS)

268 $(SLIB_FILE): $(LIB_OBJS)
269         $(QUIET_LINK)$(CC) -Wl,-soname,$@ -shared -o $@ $(LIB_OBJS) $(LDFLAGS)

271 check_list_local.h:
272         touch check_list_local.h

274 smatch.o: smatch.c $(LIB_H) smatch.h check_list.h check_list_local.h
275         $(CC) $(CFLAGS) -c smatch.c -DSMATCHDATADIR='"$(SMATCHDATADIR)"'
276 $(SMATCH_CHECKS): smatch.h smatch_slist.h smatch_extra.h avl.h
277 DEP_FILES := $(wildcard .*.o.d)

279 ifneq ($(DEP_FILES),)
280 include $(DEP_FILES)
281 endif

283 c2xml.o c2xml.sc: PKG_CFLAGS += $(LIBXML_CFLAGS)

285 pre-process.sc: CHECKER_FLAGS += -Wno-vla

287 %.o: %.c $(LIB_H)
288         $(QUIET_CC)$(CC) -o $@ -c $(ALL_CFLAGS) $<

290 %.sc: %.c sparse
291         $(QUIET_CHECK) $(CHECKER) $(CHECKER_FLAGS) -c $(ALL_CFLAGS) $<

293 ALL_OBJS :=  $(LIB_OBJS) $(foreach p,$(PROGRAMS),$(p).o $($(p)_EXTRA_DEPS))
294 selfcheck: $(ALL_OBJS:.o=.sc)


297 clean: clean-check
298         rm -f *.[oa] .*.d *.so cwchash/*.o cwchash/.*.d cwchash/tester \
299                 $(PROGRAMS) $(SLIB_FILE) pre-process.h sparse.pc version.h

301 dist:
302         @if test "$(SPARSE_VERSION)" != "v$(VERSION)" ; then \
303                 echo 'Update VERSION in the Makefile before running "make dist".
304                 exit 1 ; \
305         fi
306         git archive --format=tar --prefix=sparse-$(VERSION)/ HEAD^{tree} | gzip

308 check: all
309         $(Q)cd validation && ./test-suite

311 clean-check:
312         find validation/ \( -name "*.c.output.expected" \
313                         -o -name "*.c.output.got" \
314                         -o -name "*.c.output.diff" \
315                         -o -name "*.c.error.expected" \
316                         -o -name "*.c.error.got" \
317                         -o -name "*.c.error.diff" \
318                         \) -exec rm {} \;
```

```
**********************************************************
     139 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/README
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 There are some documents under the Documentation/ directory.

   3 For parsing implicit dependencies, see smatch_scripts/implicit_dependencies.
```

```
**********************************************************
    4695 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/allocate.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * allocate.c - simple space-efficient blob allocator.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * Simple allocator for data that doesn't get partially free'd.
 * The tokenizer and parser allocate a _lot_ of small data structures
 * (often just two-three bytes for things like small integers),
 * and since they all depend on each other you can't free them
 * individually _anyway_. So do something that is very space-
 * efficient: allocate larger "blobs", and give out individual
 * small bits and pieces of it with no maintenance overhead.
 */
#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>

#include "lib.h"
#include "allocate.h"
#include "compat.h"
#include "token.h"
#include "symbol.h"
#include "scope.h"
#include "expression.h"
#include "linearize.h"

void protect_allocations(struct allocator_struct *desc)
{
        desc->blobs = NULL;
}

void drop_all_allocations(struct allocator_struct *desc)
{
        struct allocation_blob *blob = desc->blobs;

        desc->blobs = NULL;
        desc->allocations = 0;
        desc->total_bytes = 0;
        desc->useful_bytes = 0;
        desc->freelist = NULL;
        while (blob) {
```



```
                struct allocation_blob *next = blob->next;
                blob_free(blob, desc->chunking);
                blob = next;
        }
}

void free_one_entry(struct allocator_struct *desc, void *entry)
{
        void **p = entry;
        *p = desc->freelist;
        desc->freelist = p;
}

void *allocate(struct allocator_struct *desc, unsigned int size)
{
        unsigned long alignment = desc->alignment;
        struct allocation_blob *blob = desc->blobs;
        void *retval;

        /*
         * NOTE! The freelist only works with things that are
         *  (a) sufficiently aligned
         *  (b) use a constant size
         * Don't try to free allocators that don't follow
         * these rules.
         */
        if (desc->freelist) {
                void **p = desc->freelist;
                retval = p;
                desc->freelist = *p;
                do {
                        *p = NULL;
                        p++;
                } while ((size -= sizeof(void *)) > 0);
                return retval;
        }

        desc->allocations++;
        desc->useful_bytes += size;
        size = (size + alignment - 1) & ~(alignment-1);
        if (!blob || blob->left < size) {
                unsigned int offset, chunking = desc->chunking;
                struct allocation_blob *newblob = blob_alloc(chunking);
                if (!newblob)
                        die("out of memory");
                desc->total_bytes += chunking;
                newblob->next = blob;
                blob = newblob;
                desc->blobs = newblob;
                offset = offsetof(struct allocation_blob, data);
                offset = (offset + alignment - 1) & ~(alignment-1);
                blob->left = chunking - offset;
                blob->offset = offset - offsetof(struct allocation_blob, data);
        }
        retval = blob->data + blob->offset;
        blob->offset += size;
        blob->left -= size;
        return retval;
}

void show_allocations(struct allocator_struct *x)
{
        fprintf(stderr, "%s: %lu allocations, %lu bytes (%lu total bytes, "
                        "%6.2f%% usage, %6.2f average size)\n",
                x->name, x->allocations, x->useful_bytes, x->total_bytes,
                100 * (double) x->useful_bytes / x->total_bytes,
```

```
127                 (double) x->useful_bytes / x->allocations);
128 }

130 void get_allocator_stats(struct allocator_struct *x, struct allocator_stats *s)
131 {
132         s->name = x->name;
133         s->allocations = x->allocations;
134         s->useful_bytes = x->useful_bytes;
135         s->total_bytes = x->total_bytes;
136 }

138 ALLOCATOR(ident, "identifiers");
139 ALLOCATOR(token, "tokens");
140 ALLOCATOR(context, "contexts");
141 ALLOCATOR(symbol, "symbols");
142 ALLOCATOR(expression, "expressions");
143 ALLOCATOR(statement, "statements");
144 ALLOCATOR(string, "strings");
145 ALLOCATOR(scope, "scopes");
146 __DO_ALLOCATOR(void, 0, 1, "bytes", bytes);
147 ALLOCATOR(basic_block, "basic_block");
148 ALLOCATOR(entrypoint, "entrypoint");
149 ALLOCATOR(instruction, "instruction");
150 ALLOCATOR(multijmp, "multijmp");
151 ALLOCATOR(pseudo, "pseudo");
```

```
**********************************************************
    2698 Fri Dec 21 14:59:59 2018
new/usr/src/tools/smatch/src/allocate.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef ALLOCATE_H
#define ALLOCATE_H

struct allocation_blob {
        struct allocation_blob *next;
        unsigned int left, offset;
        unsigned char data[];
};

struct allocator_struct {
        const char *name;
        struct allocation_blob *blobs;
        unsigned int alignment;
        unsigned int chunking;
        void *freelist;
        /* statistics */
        unsigned long allocations, total_bytes, useful_bytes;
};

struct allocator_stats {
        const char *name;
        unsigned int allocations;
        unsigned long total_bytes, useful_bytes;
};

extern void protect_allocations(struct allocator_struct *desc);
extern void drop_all_allocations(struct allocator_struct *desc);
extern void *allocate(struct allocator_struct *desc, unsigned int size);
extern void free_one_entry(struct allocator_struct *desc, void *entry);
extern void show_allocations(struct allocator_struct *);
extern void get_allocator_stats(struct allocator_struct *, struct allocator_stat
extern void show_allocation_stats(void);

#define __DECLARE_ALLOCATOR(type, x)            \
        extern type *__alloc_##x(int);          \
        extern void __free_##x(type *);         \
        extern void show_##x##_alloc(void);     \
        extern void get_##x##_stats(struct allocator_stats *);          \
        extern void clear_##x##_alloc(void);    \
        extern void protect_##x##_alloc(void);
#define DECLARE_ALLOCATOR(x) __DECLARE_ALLOCATOR(struct x, x)

#define __DO_ALLOCATOR(type, objsize, objalign, objname, x)     \
        static struct allocator_struct x##_allocator = {        \
                .name = objname,                                \
                .alignment = objalign,                          \
                .chunking = CHUNK };                            \
        type *__alloc_##x(int extra)                            \
        {                                                       \
                return allocate(&x##_allocator, objsize+extra); \
        }                                                       \
        void __free_##x(type *entry)                            \
        {                                                       \
                free_one_entry(&x##_allocator, entry);          \
        }                                                       \
        void show_##x##_alloc(void)                             \
        {                                                       \
                show_allocations(&x##_allocator);               \
        }                                                       \
        void get_##x##_stats(struct allocator_stats *s)         \
```



```
        {                                                       \
                get_allocator_stats(&x##_allocator, s);         \
        }                                                       \
        void clear_##x##_alloc(void)                            \
        {                                                       \
                drop_all_allocations(&x##_allocator);           \
        }                                                       \
        void protect_##x##_alloc(void)                          \
        {                                                       \
                protect_allocations(&x##_allocator);            \
        }

#define __ALLOCATOR(t, n, x)                                    \
        __DO_ALLOCATOR(t, sizeof(t), __alignof__(t), n, x)

#define ALLOCATOR(x, n) __ALLOCATOR(struct x, n, x)

DECLARE_ALLOCATOR(ident);
DECLARE_ALLOCATOR(token);
DECLARE_ALLOCATOR(context);
DECLARE_ALLOCATOR(symbol);
DECLARE_ALLOCATOR(expression);
DECLARE_ALLOCATOR(statement);
DECLARE_ALLOCATOR(string);
DECLARE_ALLOCATOR(scope);
__DECLARE_ALLOCATOR(void, bytes);
DECLARE_ALLOCATOR(basic_block);
DECLARE_ALLOCATOR(entrypoint);
DECLARE_ALLOCATOR(instruction);
DECLARE_ALLOCATOR(multijmp);
DECLARE_ALLOCATOR(pseudo);
DECLARE_ALLOCATOR(attribute);

#endif
```

```
**********************************************************
    6924 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/ast-inspect.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "ast-inspect.h"
#include "expression.h"

static inline void inspect_ptr_list(AstNode *node, const char *name, void (*insp
{
        struct ptr_list *ptrlist = node->ptr;
        void *ptr;
        int i = 0;

        node->text = g_strdup_printf("%s %s:", node->text, name);
        FOR_EACH_PTR(ptrlist, ptr) {
                char *index = g_strdup_printf("%d: ", i++);
                ast_append_child(node, index, ptr, inspect);
        } END_FOR_EACH_PTR(ptr);
}


static const char *statement_type_name(enum statement_type type)
{
        static const char *statement_type_name[] = {
                [STMT_NONE] = "STMT_NONE",
                [STMT_DECLARATION] = "STMT_DECLARATION",
                [STMT_EXPRESSION] = "STMT_EXPRESSION",
                [STMT_COMPOUND] = "STMT_COMPOUND",
                [STMT_IF] = "STMT_IF",
                [STMT_RETURN] = "STMT_RETURN",
                [STMT_CASE] = "STMT_CASE",
                [STMT_SWITCH] = "STMT_SWITCH",
                [STMT_ITERATOR] = "STMT_ITERATOR",
                [STMT_LABEL] = "STMT_LABEL",
                [STMT_GOTO] = "STMT_GOTO",
                [STMT_ASM] = "STMT_ASM",
                [STMT_CONTEXT] = "STMT_CONTEXT",
                [STMT_RANGE] = "STMT_RANGE",
        };
        return statement_type_name[type] ?: "UNKNOWN_STATEMENT_TYPE";
}

void inspect_statement(AstNode *node)
{
        struct statement *stmt = node->ptr;
        node->text = g_strdup_printf("%s %s:", node->text, statement_type_name(s
        switch (stmt->type) {
                case STMT_COMPOUND:
                        ast_append_child(node, "stmts:", stmt->stmts, inspect_st
                        break;
                case STMT_EXPRESSION:
                        ast_append_child(node, "expression:", stmt->expression,
                        break;
                case STMT_IF:
                        ast_append_child(node, "conditional:", stmt->if_conditio
                        ast_append_child(node, "if_true:", stmt->if_true, inspec
                        ast_append_child(node, "if_false:", stmt->if_false, insp
                        break;
                case STMT_ITERATOR:
                        ast_append_child(node, "break:", stmt->iterator_break, i
```



```
                        ast_append_child(node, "continue:", stmt->iterator_conti
                        ast_append_child(node, "pre_statement:", stmt->iterator_
                                        inspect_statement);
                        ast_append_child(node, "statement:", stmt->iterator_stat
                                        inspect_statement);
                        ast_append_child(node, "post_statement:", stmt->iterator
                                        inspect_statement);
                        break;

                case STMT_SWITCH:
                        ast_append_child(node, "switch_expression:", stmt->switc
                        ast_append_child(node, "switch_statement:", stmt->switch
                        ast_append_child(node, "switch_break:", stmt->switch_bre
                        ast_append_child(node, "switch_case:", stmt->switch_case
                        break;
                case STMT_CASE:
                        ast_append_child(node, "case_expression:", stmt->case_ex
                        ast_append_child(node, "case_to:", stmt->case_to, inspec
                        ast_append_child(node, "case_statement:", stmt->case_sta
                        ast_append_child(node, "case_label:", stmt->case_label,
                        break;
                case STMT_RETURN:
                        ast_append_child(node, "ret_value:", stmt->ret_value, in
                        ast_append_child(node, "ret_target:", stmt->ret_target,
                        break;

                default:
                        break;
        }
}


void inspect_statement_list(AstNode *node)
{
        inspect_ptr_list(node, "statement_list", inspect_statement);
}


static const char *symbol_type_name(enum type type)
{
        static const char *type_name[] = {
                [SYM_UNINITIALIZED] = "SYM_UNINITIALIZED",
                [SYM_PREPROCESSOR] = "SYM_PREPROCESSOR",
                [SYM_BASETYPE] = "SYM_BASETYPE",
                [SYM_NODE] = "SYM_NODE",
                [SYM_PTR] = "SYM_PTR",
                [SYM_FN] = "SYM_FN",
                [SYM_ARRAY] = "SYM_ARRAY",
                [SYM_STRUCT] = "SYM_STRUCT",
                [SYM_UNION] = "SYM_UNION",
                [SYM_ENUM] = "SYM_ENUM",
                [SYM_TYPEDEF] = "SYM_TYPEDEF",
                [SYM_TYPEOF] = "SYM_TYPEOF",
                [SYM_MEMBER] = "SYM_MEMBER",
                [SYM_BITFIELD] = "SYM_BITFIELD",
                [SYM_LABEL] = "SYM_LABEL",
                [SYM_RESTRICT] = "SYM_RESTRICT",
                [SYM_FOULED] = "SYM_FOULED",
                [SYM_KEYWORD] = "SYM_KEYWORD",
                [SYM_BAD] = "SYM_BAD",
        };
        return type_name[type] ?: "UNKNOWN_TYPE";
}


void inspect_symbol(AstNode *node)
```

```
{
        struct symbol *sym = node->ptr;
        node->text = g_strdup_printf("%s %s: %s", node->text, symbol_type_name(s
                                     builtin_typename(sym) ?: show_ident(sym->i
        ast_append_child(node, "ctype.base_type:", sym->ctype.base_type,inspect_

        switch (sym->namespace) {
                case NS_PREPROCESSOR:
                        break;
                default:
                        ast_append_child(node, "arguments:", sym->arguments, ins
                        ast_append_child(node, "symbol_list:", sym->symbol_list,
                        ast_append_child(node, "stmt:", sym->stmt, inspect_state
                        break;
        }
}


void inspect_symbol_list(AstNode *node)
{
        inspect_ptr_list(node, "symbol_list", inspect_symbol);
}


static const char *expression_type_name(enum expression_type type)
{
        static const char *expression_type_name[] = {
                [EXPR_VALUE] = "EXPR_VALUE",
                [EXPR_STRING] = "EXPR_STRING",
                [EXPR_SYMBOL] = "EXPR_SYMBOL",
                [EXPR_TYPE] = "EXPR_TYPE",
                [EXPR_BINOP] = "EXPR_BINOP",
                [EXPR_ASSIGNMENT] = "EXPR_ASSIGNMENT",
                [EXPR_LOGICAL] = "EXPR_LOGICAL",
                [EXPR_DEREF] = "EXPR_DEREF",
                [EXPR_PREOP] = "EXPR_PREOP",
                [EXPR_POSTOP] = "EXPR_POSTOP",
                [EXPR_CAST] = "EXPR_CAST",
                [EXPR_FORCE_CAST] = "EXPR_FORCE_CAST",
                [EXPR_IMPLIED_CAST] = "EXPR_IMPLIED_CAST",
                [EXPR_SIZEOF] = "EXPR_SIZEOF",
                [EXPR_ALIGNOF] = "EXPR_ALIGNOF",
                [EXPR_PTRSIZEOF] = "EXPR_PTRSIZEOF",
                [EXPR_CONDITIONAL] = "EXPR_CONDITIONAL",
                [EXPR_SELECT] = "EXPR_SELECT",
                [EXPR_STATEMENT] = "EXPR_STATEMENT",
                [EXPR_CALL] = "EXPR_CALL",
                [EXPR_COMMA] = "EXPR_COMMA",
                [EXPR_COMPARE] = "EXPR_COMPARE",
                [EXPR_LABEL] = "EXPR_LABEL",
                [EXPR_INITIALIZER] = "EXPR_INITIALIZER",
                [EXPR_IDENTIFIER] = "EXPR_IDENTIFIER",
                [EXPR_INDEX] = "EXPR_INDEX",
                [EXPR_POS] = "EXPR_POS",
                [EXPR_FVALUE] = "EXPR_FVALUE",
                [EXPR_SLICE] = "EXPR_SLICE",
                [EXPR_OFFSETOF] = "EXPR_OFFSETOF",
        };
        return expression_type_name[type] ?: "UNKNOWN_EXPRESSION_TYPE";
}

void inspect_expression(AstNode *node)
{
        struct expression *expr = node->ptr;
        node->text = g_strdup_printf("%s %s", node->text, expression_type_name(e
        switch (expr->type) {
```



```
                case EXPR_STATEMENT:
                        ast_append_child(node, "statement:", expr->statement, in
                        break;
                case EXPR_BINOP:
                case EXPR_COMMA:
                case EXPR_COMPARE:
                case EXPR_LOGICAL:
                case EXPR_ASSIGNMENT:
                        ast_append_child(node, "left:", expr->left, inspect_expr
                        ast_append_child(node, "right:", expr->right, inspect_ex
                        break;

                case EXPR_CAST:
                case EXPR_FORCE_CAST:
                case EXPR_IMPLIED_CAST:
                        ast_append_child(node, "cast_type:", expr->cast_type, in
                        ast_append_child(node, "cast_expression:", expr->cast_ex
                        break;

                case EXPR_PREOP:
                        ast_append_child(node, "unop:", expr->unop, inspect_expr
                        break;

                default:
                        break;
        }
}
```

```
**********************************************************
     330 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/ast-inspect.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 2 #ifndef _AST_INSPECT_H_
 3 #define _AST_INSPECT_H_

 5 #include "ast-model.h"

 7 void inspect_symbol(AstNode *node);
 8 void inspect_symbol_list(AstNode *node);

10 void inspect_statement(AstNode *node);
11 void inspect_statement_list(AstNode *node);

13 void inspect_expression(AstNode *node);
14 void inspect_expression_list(AstNode *node);


17 #endif
```

```
**********************************************************
   14722 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/ast-model.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 *   ast-model.c
 *
 *   A custom tree model to simplify viewing of AST objects.
 *   Modify from the Gtk+ tree view tutorial, custom-list.c
 *   by Tim-Philipp Mueller < tim at centricular dot net >
 *
 *   Copyright (C) 2010 Christopher Li
 */


#include "ast-model.h"
#include "stdint.h"

/* boring declarations of local functions */

static void ast_init(AstNode *pkg_tree);
static void ast_class_init(AstNodeClass *klass);
static void ast_tree_model_init(GtkTreeModelIface *iface);
static void ast_finalize(GObject *object);
static GtkTreeModelFlags ast_get_flags(GtkTreeModel *tree_model);
static gint ast_get_n_columns(GtkTreeModel *tree_model);
static GType ast_get_column_type(GtkTreeModel *tree_model, gint index);
static gboolean ast_get_iter(GtkTreeModel *tree_model, GtkTreeIter *iter,
                             GtkTreePath *path);
static GtkTreePath *ast_get_path(GtkTreeModel *tree_model, GtkTreeIter *iter);
static void ast_get_value(GtkTreeModel *tree_model, GtkTreeIter *iter,
                          gint column, GValue *value);
static gboolean ast_iter_next(GtkTreeModel *tree_model, GtkTreeIter *iter);
static gboolean ast_iter_children(GtkTreeModel *tree_model,
                                  GtkTreeIter *iter,
                                  GtkTreeIter *parent);
static gboolean ast_iter_has_child(GtkTreeModel *tree_model, GtkTreeIter *iter);
static gint ast_iter_n_children (GtkTreeModel *tree_model, GtkTreeIter *iter);
static gboolean ast_iter_nth_child(GtkTreeModel *tree_model, GtkTreeIter *iter,
                                   GtkTreeIter *parent, gint n);
static gboolean ast_iter_parent(GtkTreeModel *tree_model,
                                GtkTreeIter *iter,
                                GtkTreeIter *child);

static GObjectClass *parent_class = NULL;  /* GObject stuff - nothing to worry a

static inline
void inspect_child_node(AstNode *node)
{
        if (node->inspect) {
                node->inspect(node);
                node->inspect = NULL;
        }
}


static inline
AstNode* ast_nth_child(AstNode *node, int n)
{
        if (!node)
                return NULL;

        inspect_child_node(node);
```



```
        if (n >= node->childnodes->len)
                return NULL;
        return g_array_index(node->childnodes, AstNode *, n);
}


static inline
gboolean ast_set_iter(GtkTreeIter *iter, AstNode *node)
{
        iter->user_data = node;
        iter->user_data2 = iter->user_data3 = NULL;
        return node != NULL;
}


/*****************************************************************************
 *
 *  ast_get_type: here we register our new type and its interfaces
 *                with the type system. If you want to implement
 *                additional interfaces like GtkTreeSortable, you
 *                will need to do it here.
 *
 *****************************************************************************/

GType
ast_get_type (void)
{
        static GType ast_type = 0;
        static const GTypeInfo ast_info = {
                sizeof (AstNodeClass),
                NULL,                                           /* base_init */
                NULL,                                           /* base_finalize *
                (GClassInitFunc) ast_class_init,
                NULL,                                           /* class finalize
                NULL,                                           /* class_data */
                sizeof (AstNode),
                0,                                             /* n_preallocs */
                (GInstanceInitFunc) ast_init
        };
        static const GInterfaceInfo tree_model_info = {
                (GInterfaceInitFunc) ast_tree_model_init,
                NULL,
                NULL
        };



        if (ast_type)
                return ast_type;

        /* Some boilerplate type registration stuff */
        ast_type = g_type_register_static(G_TYPE_OBJECT, "AstNode",
                                          &ast_info, (GTypeFlags)0);

        /* Here we register our GtkTreeModel interface with the type system */
        g_type_add_interface_static(ast_type, GTK_TYPE_TREE_MODEL, &tree_model_i

        return ast_type;
}


/*****************************************************************************
 *
 *  ast_class_init: more boilerplate GObject/GType stuff.
 *                  Init callback for the type system,
 *                  called once when our new class is created.
```

```
 *
 ******************************************************************************/

static void
ast_class_init (AstNodeClass *klass)
{
        GObjectClass *object_class;

        parent_class = (GObjectClass*) g_type_class_peek_parent (klass);
        object_class = (GObjectClass*) klass;

        object_class->finalize = ast_finalize;
}

/******************************************************************************
 *
 *  ast_tree_model_init: init callback for the interface registration
 *                       in ast_get_type. Here we override
 *                       the GtkTreeModel interface functions that
 *                       we implement.
 *
 ******************************************************************************/

static void
ast_tree_model_init (GtkTreeModelIface *iface)
{
        iface->get_flags       = ast_get_flags;
        iface->get_n_columns   = ast_get_n_columns;
        iface->get_column_type = ast_get_column_type;
        iface->get_iter        = ast_get_iter;
        iface->get_path        = ast_get_path;
        iface->get_value       = ast_get_value;
        iface->iter_next       = ast_iter_next;
        iface->iter_children   = ast_iter_children;
        iface->iter_has_child  = ast_iter_has_child;
        iface->iter_n_children = ast_iter_n_children;
        iface->iter_nth_child  = ast_iter_nth_child;
        iface->iter_parent     = ast_iter_parent;
}


/******************************************************************************
 *
 *  ast_init: this is called every time a new ast node object
 *            instance is created (we do that in ast_new).
 *            Initialise the list structure's fields here.
 *
 ******************************************************************************/

static void
ast_init (AstNode *node)
{
        node->childnodes = g_array_new(FALSE, TRUE, sizeof(AstNode *));
        node->stamp    = g_random_int(); /* Random int to check whether iters be
}


/******************************************************************************
 *
 *  ast_finalize: this is called just before an ast node is
 *                destroyed. Free dynamically allocated memory here.
 *
 ******************************************************************************/

static void
ast_finalize (GObject *object)
```



```
{
        /*  AstNode *node = AST_NODE(object); */

        /* FIXME: free all node memory */

        /* must chain up - finalize parent */
        (* parent_class->finalize) (object);
}


/******************************************************************************
 *
 *  ast_get_flags: tells the rest of the world whether our tree model
 *                 has any special characteristics. In our case,
 *                 we have a list model (instead of a tree), and each
 *                 tree iter is valid as long as the row in question
 *                 exists, as it only contains a pointer to our struct.
 *
 ******************************************************************************/

static GtkTreeModelFlags
ast_get_flags(GtkTreeModel *tree_model)
{
        return (GTK_TREE_MODEL_ITERS_PERSIST);
}


/******************************************************************************
 *
 *  ast_get_n_columns: tells the rest of the world how many data
 *                          columns we export via the tree model interface
 *
 ******************************************************************************/

static gint
ast_get_n_columns(GtkTreeModel *tree_model)
{
        return 1;
}


/******************************************************************************
 *
 *  ast_get_column_type: tells the rest of the world which type of
 *                       data an exported model column contains
 *
 ******************************************************************************/

static GType
ast_get_column_type(GtkTreeModel *tree_model,
                        gint index)
{
        return G_TYPE_STRING;
}


/******************************************************************************
 *
 *  ast_get_iter: converts a tree path (physical position) into a
 *                tree iter structure (the content of the iter
 *                fields will only be used internally by our model).
 *                We simply store a pointer to our AstNodeItem
 *                structure that represents that row in the tree iter.
 *
 ******************************************************************************/
```

```
259 static gboolean
260 ast_get_iter(GtkTreeModel *tree_model,
261                  GtkTreeIter  *iter,
262                  GtkTreePath  *path)
263 {
264         AstNode    *node;
265         gint          *indices, depth;
266         int i;
267
268         node = AST_NODE(tree_model);
269         indices = gtk_tree_path_get_indices(path);
270         depth   = gtk_tree_path_get_depth(path);
271
272         for (i = 0; i < depth; i++)
273                 node = ast_nth_child(node, indices[i]);
274
275         return ast_set_iter(iter, node);
276 }
277
278 /*****************************************************************************
279  *
280  *  ast_get_path: converts a tree iter into a tree path (ie. the
281  *                physical position of that row in the list).
282  *
283  *****************************************************************************/
284
285 static GtkTreePath *
286 ast_get_path(GtkTreeModel *tree_model,
287                  GtkTreeIter  *iter)
288 {
289         GtkTreePath  *path;
290         AstNode   *root = AST_NODE(tree_model);
291         AstNode   *node = AST_NODE(iter->user_data);
292
293         path = gtk_tree_path_new();
294         while (node != root) {
295                 gtk_tree_path_prepend_index(path, node->index);
296                 node = node->parent;
297         }
298         return path;
299 }
300
301 /*****************************************************************************
302  *
303  *  ast_get_value: Returns a row's exported data columns
304  *                 (_get_value is what gtk_tree_model_get uses)
305  *
306  *****************************************************************************/
307
308 static void
309 ast_get_value(GtkTreeModel *tree_model,
310                   GtkTreeIter  *iter,
311                   gint          column,
312                   GValue       *value)
313 {
314         AstNode    *node = iter->user_data;
315
316         g_assert(AST_IS_NODE(tree_model));
317         if (column != 1)
318                 return;
319
320         inspect_child_node(node);
321
322         g_value_init(value, G_TYPE_STRING);
```



```
323         g_value_set_string(value, node->text);
324         return;
325 }
326
327 /*****************************************************************************
328  *
329  *  ast_iter_next: Takes an iter structure and sets it to point
330  *                 to the next row.
331  *
332  *****************************************************************************/
333
334 static gboolean
335 ast_iter_next(GtkTreeModel  *tree_model,
336                   GtkTreeIter   *iter)
337 {
338         AstNode    *node = iter->user_data;
339
340         g_assert(AST_IS_NODE (tree_model));
341
342         node = ast_nth_child(node->parent, node->index + 1);
343         return ast_set_iter(iter, node);
344 }
345
346 /*****************************************************************************
347  *
348  *  ast_iter_children: Returns TRUE or FALSE depending on whether
349  *                     the row specified by 'parent' has any children.
350  *                     If it has children, then 'iter' is set to
351  *                     point to the first child. Special case: if
352  *                     'parent' is NULL, then the first top-level
353  *                     row should be returned if it exists.
354  *
355  *****************************************************************************/
356
357 static gboolean
358 ast_iter_children(GtkTreeModel *tree_model,
359                       GtkTreeIter  *iter,
360                       GtkTreeIter  *parent)
361 {
362         return ast_iter_nth_child(tree_model, iter, parent, 0);
363 }
364
365 /*****************************************************************************
366  *
367  *  ast_iter_has_child: Returns TRUE or FALSE depending on whether
368  *                      the row specified by 'iter' has any children.
369  *                      We only have a list and thus no children.
370  *
371  *****************************************************************************/
372
373 static gboolean
374 ast_iter_has_child (GtkTreeModel *tree_model,
375                         GtkTreeIter  *iter)
376 {
377         AstNode    *node = iter->user_data;
378         inspect_child_node(node);
379         return node->childnodes->len > 0;
380 }
381
382 /*****************************************************************************
383  *
384  *  ast_iter_n_children: Returns the number of children the row
```

```
391  *                      specified by 'iter' has. This is usually 0,
392  *                      as we only have a list and thus do not have
393  *                      any children to any rows. A special case is
394  *                      when 'iter' is NULL, in which case we need
395  *                      to return the number of top-level node,
396  *                      ie. the number of rows in our list.
397  *
398  ******************************************************************************/

400 static gint
401 ast_iter_n_children (GtkTreeModel *tree_model,
402                      GtkTreeIter  *iter)
403 {
404         AstNode  *node = iter ? iter->user_data
405                               : AST_NODE(tree_model);

407         inspect_child_node(node);
408         return node->childnodes->len;
409 }


412 /******************************************************************************
413  *
414  *  ast_iter_nth_child: If the row specified by 'parent' has any
415  *                      children, set 'iter' to the n-th child and
416  *                      return TRUE if it exists, otherwise FALSE.
417  *                      A special case is when 'parent' is NULL, in
418  *                      which case we need to set 'iter' to the n-th
419  *                      row if it exists.
420  *
421  ******************************************************************************/

423 static gboolean
424 ast_iter_nth_child(GtkTreeModel *tree_model,
425                      GtkTreeIter  *iter,
426                      GtkTreeIter  *parent,
427                      gint          n)
428 {
429         AstNode    *node = parent ? parent->user_data : (AstNode*) tree_model;
430         GArray *array = node->childnodes;
431         if (n >= array->len)
432                 return FALSE;
433         iter->user_data = g_array_index(array, AstNode *, n);
434         return TRUE;
435 }


438 /******************************************************************************
439  *
440  *  ast_iter_parent: Point 'iter' to the parent node of 'child'. As
441  *                   we have a list and thus no children and no
442  *                   parents of children, we can just return FALSE.
443  *
444  ******************************************************************************/

446 static gboolean
447 ast_iter_parent (GtkTreeModel *tree_model,
448                      GtkTreeIter  *iter,
449                      GtkTreeIter  *child)
450 {
451         AstNode *node = (AstNode *) child->user_data;
452         iter->user_data = node->parent;
453         return node->parent != NULL;
454 }
```



```
457 AstNode *
458 ast_new (AstNode *parent, int index, const char *text, void *ptr, void (*inspect
459 {
460         AstNode *node = (AstNode*) g_object_new (AST_TYPE_NODE, NULL);
461         g_assert(node != NULL);
462         node->parent = parent;
463         node->index = index;
464         node->text = text;
465         node->inspect = inspect;
466         node->ptr = ptr;
467         return node;
468 }
```

```
**********************************************************
    2246 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/ast-model.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * ast-model.h
 *
 * Copyright (C) 2010 Christopher Li.
 *
 */

#ifndef _ast_model_h_
#define _ast_model_h_

#include <stdint.h>
#include <gtk/gtk.h>
#include "lib.h"

#define AST_TYPE_NODE                   (ast_get_type ())
#define AST_NODE(obj)                   (G_TYPE_CHECK_INSTANCE_CAST ((obj), AST_T
#define AST_NODE_CLASS(klass)           (G_TYPE_CHECK_CLASS_CAST ((klass),  AST_T
#define AST_IS_NODE(obj)                (G_TYPE_CHECK_INSTANCE_TYPE ((obj), AST_T
#define AST_IS_NODE_CLASS(klass)        (G_TYPE_CHECK_CLASS_TYPE ((klass),  AST_T
#define AST_NODE_GET_CLASS(obj)         (G_TYPE_INSTANCE_GET_CLASS ((obj),  AST_T

enum
{
        AST_COL_RECORD = 0,
        AST_COL_NAME,
        AST_N_COLUMNS,
} ;


typedef struct AstNode AstNode;
typedef struct AstNodeClass AstNodeClass;



/* AstNode: this structure contains everything we need for our
 *             model implementation. You can add extra fields to
 *             this structure, e.g. hashtables to quickly lookup
 *             rows or whatever else you might need, but it is
 *             crucial that 'parent' is the first member of the
 *             structure.                                      */

struct AstNode
{
        GObject         base;      /* this MUST be the first member */

        AstNode *parent;
        int index;
        const gchar *text;
        void (*inspect)(struct AstNode* node);
        void *ptr;
        GArray *childnodes;
        gint stamp;
};



/* AstNodeClass: more boilerplate GObject stuff */

struct AstNodeClass
```



```
{
        GObjectClass base_class;
};


GType ast_get_type(void);
AstNode* ast_new(AstNode *parent, int index, const char *prefix, void *ptr, void


static inline
AstNode* ast_append_child(AstNode *parent, const char *text,
                          void *ptr, void (*inspect)(AstNode*))
{
        if (ptr) {
                AstNode *child = ast_new(parent, parent->childnodes->len,
                                         text, ptr, inspect);
                g_array_append_val(parent->childnodes, child);
                return child;
        }
        return NULL;
}

static inline
void ast_append_attribute(AstNode *parent, const char *text)
{
        AstNode *child = ast_new(parent, parent->childnodes->len, text, NULL, NU
        g_array_append_val(parent->childnodes, child);
}

#endif /* _ast_h_*/
```

```
**********************************************************
    1165 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/ast-view.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 2 #include <stdlib.h>
 3 #include "ast-model.h"
 4 #include "ast-inspect.h"
 5 #include "ast-view.h"

 7 static GtkWidget *
 8 create_view_and_model (void *ptr)
 9 {
10         GtkTreeViewColumn   *text;
11         GtkCellRenderer *renderer;
12         AstNode *root;
13         GtkWidget *view;

15         root = ast_new(NULL, 0, "", ptr, inspect_symbol_list);

17         view = gtk_tree_view_new_with_model(GTK_TREE_MODEL(root));

19         g_object_unref(root); /* destroy store automatically with view */

21         renderer = gtk_cell_renderer_text_new();
22         text = gtk_tree_view_column_new_with_attributes("Node", renderer,
23                                                         "text", AST_COL_NAME,
24                                                         NULL);
25         gtk_tree_view_append_column(GTK_TREE_VIEW(view), text);

27         return view;
28 }

30 void
31 treeview_main (struct symbol_list *syms)
32 {
33         GtkWidget *window, *view, *scrollwin;

35         window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
36         gtk_window_set_default_size (GTK_WINDOW(window), 600, 800);
37         g_signal_connect(window, "delete_event", gtk_main_quit, NULL);

39         scrollwin = gtk_scrolled_window_new(NULL,NULL);

41         view = create_view_and_model(syms);

43         gtk_container_add(GTK_CONTAINER(scrollwin), view);
44         gtk_container_add(GTK_CONTAINER(window), scrollwin);

46         gtk_widget_show_all(window);

48         gtk_main();
49 }
```

```
**********************************************************
      95 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/ast-view.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  2 #include <gtk/gtk.h>
  3 #include "lib.h"

  5 extern void treeview_main(struct symbol_list *syms);
```

```
**********************************************************
   11656 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/avl.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Joseph Adams <joeyadams3.14159@gmail.com>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include <assert.h>
#include <stdlib.h>

#include "smatch.h"
#include "smatch_slist.h"

static AvlNode *mkNode(const struct sm_state *sm);
static void freeNode(AvlNode *node);

static AvlNode *lookup(const struct stree *avl, AvlNode *node, const struct sm_s

static bool insert_sm(struct stree *avl, AvlNode **p, const struct sm_state *sm)
static bool remove_sm(struct stree *avl, AvlNode **p, const struct sm_state *sm,
static bool removeExtremum(AvlNode **p, int side, AvlNode **ret);

static int sway(AvlNode **p, int sway);
static void balance(AvlNode **p, int side);

static bool checkBalances(AvlNode *node, int *height);
static bool checkOrder(struct stree *avl);
static size_t countNode(AvlNode *node);

int unfree_stree;

/*
 * Utility macros for converting between
 * "balance" values (-1 or 1) and "side" values (0 or 1).
 *
 * bal(0)   == -1
 * bal(1)   == +1
 * side(-1) == 0
 * side(+1) == 1
 */
#define bal(side) ((side) == 0 ? -1 : 1)
#define side(bal) ((bal)  == 1 ?  1 : 0)

static struct stree *avl_new(void)
{
```



```
        struct stree *avl = malloc(sizeof(*avl));

        unfree_stree++;
        assert(avl != NULL);

        avl->root = NULL;
        avl->base_stree = NULL;
        avl->has_states = calloc(num_checks + 1, sizeof(char));
        avl->count = 0;
        avl->stree_id = 0;
        avl->references = 1;
        return avl;
}

void free_stree(struct stree **avl)
{
        if (!*avl)
                return;

        assert((*avl)->references > 0);

        (*avl)->references--;
        if ((*avl)->references != 0) {
                *avl = NULL;
                return;
        }

        unfree_stree--;

        freeNode((*avl)->root);
        free(*avl);
        *avl = NULL;
}

struct sm_state *avl_lookup(const struct stree *avl, const struct sm_state *sm)
{
        AvlNode *found;

        if (!avl)
                return NULL;
        if (sm->owner != USHRT_MAX &&
            !avl->has_states[sm->owner])
                return NULL;
        found = lookup(avl, avl->root, sm);
        if (!found)
                return NULL;
        return (struct sm_state *)found->sm;
}

AvlNode *avl_lookup_node(const struct stree *avl, const struct sm_state *sm)
{
        return lookup(avl, avl->root, sm);
}

size_t stree_count(const struct stree *avl)
{
        if (!avl)
                return 0;
        return avl->count;
}

static struct stree *clone_stree_real(struct stree *orig)
{
        struct stree *new = avl_new();
        AvlIter i;
```

```
127         avl_foreach(i, orig)
128                 avl_insert(&new, i.sm);
129
130         new->base_stree = orig->base_stree;
131         return new;
132 }
133
134 bool avl_insert(struct stree **avl, const struct sm_state *sm)
135 {
136         size_t old_count;
137
138         if (!*avl)
139                 *avl = avl_new();
140         if ((*avl)->references > 1) {
141                 (*avl)->references--;
142                 *avl = clone_stree_real(*avl);
143         }
144         old_count = (*avl)->count;
145         /* fortunately we never call get_state() on "unnull_path" */
146         if (sm->owner != USHRT_MAX)
147                 (*avl)->has_states[sm->owner] = 1;
148         insert_sm(*avl, &(*avl)->root, sm);
149         return (*avl)->count != old_count;
150 }
151
152 bool avl_remove(struct stree **avl, const struct sm_state *sm)
153 {
154         AvlNode *node = NULL;
155
156         if (!*avl)
157                 return false;
158         /* it's fairly rare for smatch to call avl_remove */
159         if ((*avl)->references > 1) {
160                 (*avl)->references--;
161                 *avl = clone_stree_real(*avl);
162         }
163
164         remove_sm(*avl, &(*avl)->root, sm, &node);
165
166         if ((*avl)->count == 0)
167                 free_stree(avl);
168
169         if (node == NULL) {
170                 return false;
171         } else {
172                 free(node);
173                 return true;
174         }
175 }
176
177 static AvlNode *mkNode(const struct sm_state *sm)
178 {
179         AvlNode *node = malloc(sizeof(*node));
180
181         assert(node != NULL);
182
183         node->sm = sm;
184         node->lr[0] = NULL;
185         node->lr[1] = NULL;
186         node->balance = 0;
187         return node;
188 }
189
190 static void freeNode(AvlNode *node)
191 {
192         if (node) {
```



```
193                 freeNode(node->lr[0]);
194                 freeNode(node->lr[1]);
195                 free(node);
196         }
197 }
198
199 static AvlNode *lookup(const struct stree *avl, AvlNode *node, const struct sm_s
200 {
201         int cmp;
202
203         if (node == NULL)
204                 return NULL;
205
206         cmp = cmp_tracker(sm, node->sm);
207
208         if (cmp < 0)
209                 return lookup(avl, node->lr[0], sm);
210         if (cmp > 0)
211                 return lookup(avl, node->lr[1], sm);
212         return node;
213 }
214
215 /*
216  * Insert an sm into a subtree, rebalancing if necessary.
217  *
218  * Return true if the subtree's height increased.
219  */
220 static bool insert_sm(struct stree *avl, AvlNode **p, const struct sm_state *sm)
221 {
222         if (*p == NULL) {
223                 *p = mkNode(sm);
224                 avl->count++;
225                 return true;
226         } else {
227                 AvlNode *node = *p;
228                 int      cmp  = cmp_tracker(sm, node->sm);
229
230                 if (cmp == 0) {
231                         node->sm = sm;
232                         return false;
233                 }
234
235                 if (!insert_sm(avl, &node->lr[side(cmp)], sm))
236                         return false;
237
238                 /* If tree's balance became -1 or 1, it means the tree's height
239                 return sway(p, cmp) != 0;
240         }
241 }
242
243 /*
244  * Remove the node matching the given sm.
245  * If present, return the removed node through *ret .
246  * The returned node's lr and balance are meaningless.
247  *
248  * Return true if the subtree's height decreased.
249  */
250 static bool remove_sm(struct stree *avl, AvlNode **p, const struct sm_state *sm,
251 {
252         if (p == NULL || *p == NULL) {
253                 return false;
254         } else {
255                 AvlNode *node = *p;
256                 int      cmp  = cmp_tracker(sm, node->sm);
257
258                 if (cmp == 0) {
```

```
259                         *ret = node;
260                         avl->count--;
261
262                         if (node->lr[0] != NULL && node->lr[1] != NULL) {
263                                 AvlNode *replacement;
264                                 int      side;
265                                 bool     shrunk;
266
267                                 /* Pick a subtree to pull the replacement from s
268                                  * this node doesn't have to be rebalanced. */
269                                 side = node->balance <= 0 ? 0 : 1;
270
271                                 shrunk = removeExtremum(&node->lr[side], 1 - sid
272
273                                 replacement->lr[0]   = node->lr[0];
274                                 replacement->lr[1]   = node->lr[1];
275                                 replacement->balance = node->balance;
276                                 *p = replacement;
277
278                                 if (!shrunk)
279                                         return false;
280
281                                 replacement->balance -= bal(side);
282
283                                 /* If tree's balance became 0, it means the tree
284                                 return replacement->balance == 0;
285                         }
286
287                         if (node->lr[0] != NULL)
288                                 *p = node->lr[0];
289                         else
290                                 *p = node->lr[1];
291
292                         return true;
293
294                 } else {
295                         if (!remove_sm(avl, &node->lr[side(cmp)], sm, ret))
296                                 return false;
297
298                         /* If tree's balance became 0, it means the tree's heigh
299                         return sway(p, -cmp) == 0;
300                 }
301         }
302 }
303
304 /*
305  * Remove either the left-most (if side == 0) or right-most (if side == 1)
306  * node in a subtree, returning the removed node through *ret .
307  * The returned node's lr and balance are meaningless.
308  *
309  * The subtree must not be empty (i.e. *p must not be NULL).
310  *
311  * Return true if the subtree's height decreased.
312  */
313 static bool removeExtremum(AvlNode **p, int side, AvlNode **ret)
314 {
315         AvlNode *node = *p;
316
317         if (node->lr[side] == NULL) {
318                 *ret = node;
319                 *p = node->lr[1 - side];
320                 return true;
321         }
322
323         if (!removeExtremum(&node->lr[side], side, ret))
324                 return false;
```



```
325
326         /* If tree's balance became 0, it means the tree's height shrank due to
327         return sway(p, -bal(side)) == 0;
328 }
329
330 /*
331  * Rebalance a node if necessary.  Think of this function
332  * as a higher-level interface to balance().
333  *
334  * sway must be either -1 or 1, and indicates what was added to
335  * the balance of this node by a prior operation.
336  *
337  * Return the new balance of the subtree.
338  */
339 static int sway(AvlNode **p, int sway)
340 {
341         if ((*p)->balance != sway)
342                 (*p)->balance += sway;
343         else
344                 balance(p, side(sway));
345
346         return (*p)->balance;
347 }
348
349 /*
350  * Perform tree rotations on an unbalanced node.
351  *
352  * side == 0 means the node's balance is -2 .
353  * side == 1 means the node's balance is +2 .
354  */
355 static void balance(AvlNode **p, int side)
356 {
357         AvlNode  *node  = *p,
358                  *child = node->lr[side];
359         int opposite    = 1 - side;
360         int bal         = bal(side);
361
362         if (child->balance != -bal) {
363                 /* Left-left (side == 0) or right-right (side == 1) */
364                 node->lr[side]      = child->lr[opposite];
365                 child->lr[opposite] = node;
366                 *p = child;
367
368                 child->balance -= bal;
369                 node->balance = -child->balance;
370
371         } else {
372                 /* Left-right (side == 0) or right-left (side == 1) */
373                 AvlNode *grandchild = child->lr[opposite];
374
375                 node->lr[side]           = grandchild->lr[opposite];
376                 child->lr[opposite]      = grandchild->lr[side];
377                 grandchild->lr[side]     = child;
378                 grandchild->lr[opposite] = node;
379                 *p = grandchild;
380
381                 node->balance       = 0;
382                 child->balance      = 0;
383
384                 if (grandchild->balance == bal)
385                         node->balance  = -bal;
386                 else if (grandchild->balance == -bal)
387                         child->balance = bal;
388
389                 grandchild->balance = 0;
390         }
```

```
391 }


394 /************************ avl_check_invariants() ************************/

396 bool avl_check_invariants(struct stree *avl)
397 {
398         int    dummy;

400         return checkBalances(avl->root, &dummy)
401             && checkOrder(avl)
402             && countNode(avl->root) == avl->count;
403 }

405 static bool checkBalances(AvlNode *node, int *height)
406 {
407         if (node) {
408                 int h0, h1;

410                 if (!checkBalances(node->lr[0], &h0))
411                         return false;
412                 if (!checkBalances(node->lr[1], &h1))
413                         return false;

415                 if (node->balance != h1 - h0 || node->balance < -1 || node->bala
416                         return false;

418                 *height = (h0 > h1 ? h0 : h1) + 1;
419                 return true;
420         } else {
421                 *height = 0;
422                 return true;
423         }
424 }

426 static bool checkOrder(struct stree *avl)
427 {
428         AvlIter     i;
429         const struct sm_state *last = NULL;
430         bool        last_set = false;

432         avl_foreach(i, avl) {
433                 if (last_set && cmp_tracker(last, i.sm) >= 0)
434                         return false;
435                 last     = i.sm;
436                 last_set = true;
437         }

439         return true;
440 }

442 static size_t countNode(AvlNode *node)
443 {
444         if (node)
445                 return 1 + countNode(node->lr[0]) + countNode(node->lr[1]);
446         else
447                 return 0;
448 }


451 /************************ Traversal ************************/

453 void avl_iter_begin(AvlIter *iter, struct stree *avl, AvlDirection dir)
454 {
455         AvlNode *node;
```



```
457         iter->stack_index = 0;
458         iter->direction   = dir;

460         if (!avl || !avl->root) {
461                 iter->sm      = NULL;
462                 iter->node     = NULL;
463                 return;
464         }
465         node = avl->root;

467         while (node->lr[dir] != NULL) {
468                 iter->stack[iter->stack_index++] = node;
469                 node = node->lr[dir];
470         }

472         iter->sm   = (struct sm_state *) node->sm;
473         iter->node  = node;
474 }

476 void avl_iter_next(AvlIter *iter)
477 {
478         AvlNode     *node = iter->node;
479         AvlDirection dir  = iter->direction;

481         if (node == NULL)
482                 return;

484         node = node->lr[1 - dir];
485         if (node != NULL) {
486                 while (node->lr[dir] != NULL) {
487                         iter->stack[iter->stack_index++] = node;
488                         node = node->lr[dir];
489                 }
490         } else if (iter->stack_index > 0) {
491                 node = iter->stack[--iter->stack_index];
492         } else {
493                 iter->sm      = NULL;
494                 iter->node     = NULL;
495                 return;
496         }

498         iter->node  = node;
499         iter->sm   = (struct sm_state *) node->sm;
500 }

502 struct stree *clone_stree(struct stree *orig)
503 {
504         if (!orig)
505                 return NULL;

507         orig->references++;
508         return orig;
509 }

511 void set_stree_id(struct stree **stree, int stree_id)
512 {
513         if ((*stree)->stree_id != 0)
514                 *stree = clone_stree_real(*stree);

516         (*stree)->stree_id = stree_id;
517 }

519 int get_stree_id(struct stree *stree)
520 {
521         if (!stree)
522                 return -1;
```

```
523         return stree->stree_id;
524 }
```

```
**********************************************************
    4069 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/avl.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Joseph Adams <joeyadams3.14159@gmail.com>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#ifndef CCAN_AVL_H
#define CCAN_AVL_H

#include <stdbool.h>
#include <stddef.h>

struct sm_state;

typedef struct AvlNode       AvlNode;
typedef struct AvlIter       AvlIter;

struct stree {
        AvlNode    *root;
        struct stree *base_stree;
        char *has_states;
        size_t      count;
        int stree_id;
        int references;
};

void free_stree(struct stree **avl);
        /* Free an stree tree. */

struct sm_state *avl_lookup(const struct stree *avl, const struct sm_state *sm);
        /* O(log n). Lookup a sm.  Return NULL if the sm is not present. */

#define avl_member(avl, sm) (!!avl_lookup_node(avl, sm))
        /* O(log n). See if a sm is present. */

size_t stree_count(const struct stree *avl);
        /* O(1). Return the number of elements in the tree. */

bool avl_insert(struct stree **avl, const struct sm_state *sm);
        /*
         * O(log n). Insert an sm or replace it if already present.
         *
         * Return false if the insertion replaced an existing sm.
         */
```



```
bool avl_remove(struct stree **avl, const struct sm_state *sm);
        /*
         * O(log n). Remove an sm (if present).
         *
         * Return true if it was removed.
         */

bool avl_check_invariants(struct stree *avl);
        /* For testing purposes.  This function will always return true :-) */


/************************ Traversal ************************/

#define avl_foreach(iter, avl)         avl_traverse(iter, avl, FORWARD)
        /*
         * O(n). Traverse an stree tree in order.
         *
         * Example:
         *
         * AvlIter i;
         *
         * avl_foreach(i, avl)
         *     printf("%s -> %s\n", i.sm->name, i.sm->state->name);
         */

#define FOR_EACH_SM(avl, _sm) {                 \
        AvlIter _i;                             \
        avl_foreach(_i, avl) {                  \
                _sm = _i.sm;

#define END_FOR_EACH_SM(_sm) }}

#define FOR_EACH_MY_SM(_owner, avl, _sm) {      \
        AvlIter _i;                             \
        avl_foreach(_i, avl) {                  \
                _sm = _i.sm;                    \
                if (_sm->owner != _owner)       \
                        continue;               \

#define avl_foreach_reverse(iter, avl) avl_traverse(iter, avl, BACKWARD)
        /* O(n). Traverse an stree tree in reverse order. */

typedef enum AvlDirection {FORWARD = 0, BACKWARD = 1} AvlDirection;

struct AvlIter {
        struct sm_state *sm;
        AvlNode      *node;

        /* private */
        AvlNode      *stack[100];
        int           stack_index;
        AvlDirection  direction;
};

void avl_iter_begin(AvlIter *iter, struct stree *avl, AvlDirection dir);
void avl_iter_next(AvlIter *iter);
#define avl_traverse(iter, avl, direction)        \
        for (avl_iter_begin(&(iter), avl, direction); \
             (iter).node != NULL;                     \
             avl_iter_next(&iter))


/***************** Internal data structures *****************/

struct AvlNode {
```

```
127         const struct sm_state *sm;

129         AvlNode    *lr[2];
130         int         balance; /* -1, 0, or 1 */
131 };

133 AvlNode *avl_lookup_node(const struct stree *avl, const struct sm_state *sm);
134         /* O(log n). Lookup an stree node by sm.  Return NULL if not present. */

136 struct stree *clone_stree(struct stree *orig);

138 void set_stree_id(struct stree **stree, int id);
139 int get_stree_id(struct stree *stree);

141 #endif
```

```
**********************************************************
    1436 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/bitmap.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef BITMAP_H
#define BITMAP_H

#define BITS_IN_LONG    (sizeof(unsigned long)*8)
#define LONGS(x)        ((x + BITS_IN_LONG - 1) & -BITS_IN_LONG)

/* Every bitmap gets its own type */
#define DECLARE_BITMAP(name, x) unsigned long name[LONGS(x)]

static inline int test_bit(unsigned int nr, unsigned long *bitmap)
{
        unsigned long offset = nr / BITS_IN_LONG;
        unsigned long bit = nr & (BITS_IN_LONG-1);
        return (bitmap[offset] >> bit) & 1;
}

static inline void set_bit(unsigned int nr, unsigned long *bitmap)
{
        unsigned long offset = nr / BITS_IN_LONG;
        unsigned long bit = nr & (BITS_IN_LONG-1);
        bitmap[offset] |= 1UL << bit;
}

static inline void clear_bit(unsigned int nr, unsigned long *bitmap)
{
        unsigned long offset = nr / BITS_IN_LONG;
        unsigned long bit = nr & (BITS_IN_LONG-1);
        bitmap[offset] &= ~(1UL << bit);
}

static inline int test_and_set_bit(unsigned int nr, unsigned long *bitmap)
{
        unsigned long offset = nr / BITS_IN_LONG;
        unsigned long bit = nr & (BITS_IN_LONG-1);
        unsigned long old = bitmap[offset];
        unsigned long mask = 1UL << bit;
        bitmap[offset] = old | mask;
        return (old & mask) != 0;
}

static inline int test_and_clear_bit(unsigned int nr, unsigned long *bitmap)
{
        unsigned long offset = nr / BITS_IN_LONG;
        unsigned long bit = nr & (BITS_IN_LONG-1);
        unsigned long old = bitmap[offset];
        unsigned long mask = 1UL << bit;
        bitmap[offset] = old & ~mask;
        return (old & mask) != 0;
}

#endif /* BITMAP_H */
```

```
**********************************************************
    7007 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/builtin.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * builtin evaluation & expansion.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include "expression.h"
#include "expand.h"
#include "symbol.h"
#include "compat/bswap.h"

static int evaluate_to_int_const_expr(struct expression *expr)
{
        expr->ctype = &int_ctype;
        expr->flags |= CEF_SET_ICE;
        return 1;
}

static int evaluate_pure_unop(struct expression *expr)
{
        struct expression *arg = first_expression(expr->args);
        int flags = arg->flags;

        /*
         * Allow such functions with a constant integer expression
         * argument to be treated as a *constant* integer.
         * This allow us to use them in switch() { case ...:
         */
        flags |= (flags & CEF_ICE) ? CEF_SET_INT : 0;
        expr->flags = flags;
        return 1;
}


static int evaluate_expect(struct expression *expr)
{
        /* Should we evaluate it to return the type of the first argument? */
        expr->ctype = &int_ctype;
        return 1;
}
```



```
static int arguments_choose(struct expression *expr)
{
        struct expression_list *arglist = expr->args;
        struct expression *arg;
        int i = 0;

        FOR_EACH_PTR (arglist, arg) {
                if (!evaluate_expression(arg))
                        return 0;
                i++;
        } END_FOR_EACH_PTR(arg);
        if (i < 3) {
                sparse_error(expr->pos,
                             "not enough arguments for __builtin_choose_expr");
                return 0;
        } if (i > 3) {
                sparse_error(expr->pos,
                             "too many arguments for __builtin_choose_expr");
                return 0;
        }
        return 1;
}

static int evaluate_choose(struct expression *expr)
{
        struct expression_list *list = expr->args;
        struct expression *arg, *args[3];
        int n = 0;

        /* there will be exactly 3; we'd already verified that */
        FOR_EACH_PTR(list, arg) {
                args[n++] = arg;
        } END_FOR_EACH_PTR(arg);

        *expr = get_expression_value(args[0]) ? *args[1] : *args[2];

        return 1;
}

static int expand_expect(struct expression *expr, int cost)
{
        struct expression *arg = first_ptr_list((struct ptr_list *) expr->args);

        if (arg)
                *expr = *arg;
        return 0;
}

/*
 * __builtin_warning() has type "int" and always returns 1,
 * so that you can use it in conditionals or whatever
 */
static int expand_warning(struct expression *expr, int cost)
{
        struct expression *arg;
        struct expression_list *arglist = expr->args;

        FOR_EACH_PTR (arglist, arg) {
                /*
                 * Constant strings get printed out as a warning. By the
                 * time we get here, the EXPR_STRING has been fully
                 * evaluated, so by now it's an anonymous symbol with a
                 * string initializer.
                 *
                 * Just for the heck of it, allow any constant string
                 * symbol.
```

```
127                  */
128                 if (arg->type == EXPR_SYMBOL) {
129                         struct symbol *sym = arg->symbol;
130                         if (sym->initializer && sym->initializer->type == EXPR_S
131                                 struct string *string = sym->initializer->string
132                                 warning(expr->pos, "%*s", string->length-1, stri
133                         }
134                         continue;
135                 }
136
137                 /*
138                  * Any other argument is a conditional. If it's
139                  * non-constant, or it is false, we exit and do
140                  * not print any warning.
141                  */
142                 if (arg->type != EXPR_VALUE)
143                         goto out;
144                 if (!arg->value)
145                         goto out;
146         } END_FOR_EACH_PTR(arg);
147 out:
148         expr->type = EXPR_VALUE;
149         expr->value = 1;
150         expr->taint = 0;
151         return 0;
152 }
153
154 /* The arguments are constant if the cost of all of them is zero */
155 static int expand_constant_p(struct expression *expr, int cost)
156 {
157         expr->type = EXPR_VALUE;
158         expr->value = !cost;
159         expr->taint = 0;
160         return 0;
161 }
162
163 /* The arguments are safe, if their cost is less than SIDE_EFFECTS */
164 static int expand_safe_p(struct expression *expr, int cost)
165 {
166         expr->type = EXPR_VALUE;
167         expr->value = (cost < SIDE_EFFECTS);
168         expr->taint = 0;
169         return 0;
170 }
171
172 static struct symbol_op constant_p_op = {
173         .evaluate = evaluate_to_int_const_expr,
174         .expand = expand_constant_p
175 };
176
177 static struct symbol_op safe_p_op = {
178         .evaluate = evaluate_to_int_const_expr,
179         .expand = expand_safe_p
180 };
181
182 static struct symbol_op warning_op = {
183         .evaluate = evaluate_to_int_const_expr,
184         .expand = expand_warning
185 };
186
187 static struct symbol_op expect_op = {
188         .evaluate = evaluate_expect,
189         .expand = expand_expect
190 };
191
192 static struct symbol_op choose_op = {
```



```
193         .evaluate = evaluate_choose,
194         .args = arguments_choose,
195 };
196
197 /* The argument is constant and valid if the cost is zero */
198 static int expand_bswap(struct expression *expr, int cost)
199 {
200         struct expression *arg;
201         long long val;
202
203         if (cost)
204                 return cost;
205
206         /* the arguments number & type have already been checked */
207         arg = first_expression(expr->args);
208         val = get_expression_value_silent(arg);
209         switch (expr->ctype->bit_size) {
210         case 16: expr->value = bswap16(val); break;
211         case 32: expr->value = bswap32(val); break;
212         case 64: expr->value = bswap64(val); break;
213         default: /* impossible error */
214                 return SIDE_EFFECTS;
215         }
216
217         expr->type = EXPR_VALUE;
218         expr->taint = 0;
219         return 0;
220 }
221
222 static struct symbol_op bswap_op = {
223         .evaluate = evaluate_pure_unop,
224         .expand = expand_bswap,
225 };
226
227
228 /*
229  * Builtin functions
230  */
231 static struct symbol builtin_fn_type = { .type = SYM_FN /* , .variadic =1 */ };
232 static struct sym_init {
233         const char *name;
234         struct symbol *base_type;
235         unsigned int modifiers;
236         struct symbol_op *op;
237 } builtins_table[] = {
238         { "__builtin_constant_p", &builtin_fn_type, MOD_TOPLEVEL, &constant_p_op
239         { "__builtin_safe_p", &builtin_fn_type, MOD_TOPLEVEL, &safe_p_op },
240         { "__builtin_warning", &builtin_fn_type, MOD_TOPLEVEL, &warning_op },
241         { "__builtin_expect", &builtin_fn_type, MOD_TOPLEVEL, &expect_op },
242         { "__builtin_choose_expr", &builtin_fn_type, MOD_TOPLEVEL, &choose_op },
243         { "__builtin_bswap16", NULL, MOD_TOPLEVEL, &bswap_op },
244         { "__builtin_bswap32", NULL, MOD_TOPLEVEL, &bswap_op },
245         { "__builtin_bswap64", NULL, MOD_TOPLEVEL, &bswap_op },
246         { NULL,         NULL,           0 }
247 };
248
249 void init_builtins(int stream)
250 {
251         struct sym_init *ptr;
252
253         builtin_fn_type.variadic = 1;
254         for (ptr = builtins_table; ptr->name; ptr++) {
255                 struct symbol *sym;
256                 sym = create_symbol(stream, ptr->name, SYM_NODE, NS_SYMBOL);
257                 sym->ctype.base_type = ptr->base_type;
258                 sym->ctype.modifiers = ptr->modifiers;
```

```
259                     sym->op = ptr->op;
260             }
261     }
```

```
**********************************************************
    7854 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/c2xml.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Sparse c2xml
 *
 * Dumps the parse tree as an xml document
 *
 * Copyright (C) 2007 Rob Taylor
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <assert.h>
#include <libxml/parser.h>
#include <libxml/tree.h>

#include "expression.h"
#include "parse.h"
#include "scope.h"
#include "symbol.h"

static xmlDocPtr doc = NULL;       /* document pointer */
static xmlNodePtr root_node = NULL;/* root node pointer */
static int idcount = 0;

static void examine_symbol(struct symbol *sym, xmlNodePtr node);

static xmlAttrPtr newProp(xmlNodePtr node, const char *name, const char *value)
{
        return xmlNewProp(node, BAD_CAST name, BAD_CAST value);
}

static xmlAttrPtr newNumProp(xmlNodePtr node, const char *name, int value)
{
        char buf[256];
        snprintf(buf, 256, "%d", value);
        return newProp(node, name, buf);
}

static xmlAttrPtr newIdProp(xmlNodePtr node, const char *name, unsigned int id)
{
        char buf[256];
```



```
        snprintf(buf, 256, "_%d", id);
        return newProp(node, name, buf);
}

static xmlNodePtr new_sym_node(struct symbol *sym, const char *name, xmlNodePtr
{
        xmlNodePtr node;
        const char *ident = show_ident(sym->ident);

        assert(name != NULL);
        assert(sym != NULL);
        assert(parent != NULL);

        node = xmlNewChild(parent, NULL, BAD_CAST "symbol", NULL);

        newProp(node, "type", name);

        newIdProp(node, "id", idcount);

        if (sym->ident && ident)
                newProp(node, "ident", ident);
        newProp(node, "file", stream_name(sym->pos.stream));

        newNumProp(node, "start-line", sym->pos.line);
        newNumProp(node, "start-col", sym->pos.pos);

        if (sym->endpos.type) {
                newNumProp(node, "end-line", sym->endpos.line);
                newNumProp(node, "end-col", sym->endpos.pos);
                if (sym->pos.stream != sym->endpos.stream)
                        newProp(node, "end-file", stream_name(sym->endpos.stream
        }
        sym->aux = node;

        idcount++;

        return node;
}

static inline void examine_members(struct symbol_list *list, xmlNodePtr node)
{
        struct symbol *sym;

        FOR_EACH_PTR(list, sym) {
                examine_symbol(sym, node);
        } END_FOR_EACH_PTR(sym);
}

static void examine_modifiers(struct symbol *sym, xmlNodePtr node)
{
        const char *modifiers[] = {
                        "auto",
                        "register",
                        "static",
                        "extern",
                        "const",
                        "volatile",
                        "signed",
                        "unsigned",
                        "char",
                        "short",
                        "long",
                        "long-long",
                        "typedef",
                        NULL,
                        NULL,
```

```
                        NULL,
                        NULL,
                        NULL,
                        "inline",
                        "addressable",
                        "nocast",
                        "noderef",
                        "accessed",
                        "toplevel",
                        "label",
                        "assigned",
                        "type-type",
                        "safe",
                        "user-type",
                        "force",
                        "explicitly-signed",
                        "bitwise"};

        int i;

        if (sym->namespace != NS_SYMBOL)
                return;

        /*iterate over the 32 bit bitfield*/
        for (i=0; i < 32; i++) {
                if ((sym->ctype.modifiers & 1<<i) && modifiers[i])
                        newProp(node, modifiers[i], "1");
        }
}

static void
examine_layout(struct symbol *sym, xmlNodePtr node)
{
        examine_symbol_type(sym);

        newNumProp(node, "bit-size", sym->bit_size);
        newNumProp(node, "alignment", sym->ctype.alignment);
        newNumProp(node, "offset", sym->offset);
        if (is_bitfield_type(sym)) {
                newNumProp(node, "bit-offset", sym->bit_offset);
        }
}

static void examine_symbol(struct symbol *sym, xmlNodePtr node)
{
        xmlNodePtr child = NULL;
        const char *base;
        int array_size;

        if (!sym)
                return;
        if (sym->aux)           /*already visited */
                return;

        if (sym->ident && sym->ident->reserved)
                return;

        child = new_sym_node(sym, get_type_name(sym->type), node);
        examine_modifiers(sym, child);
        examine_layout(sym, child);

        if (sym->ctype.base_type) {
                if ((base = builtin_typename(sym->ctype.base_type)) == NULL) {
                        if (!sym->ctype.base_type->aux) {
                                examine_symbol(sym->ctype.base_type, root_node);
                        }
```



```
                        xmlNewProp(child, BAD_CAST "base-type",
                                   xmlGetProp((xmlNodePtr)sym->ctype.base_type->
                } else {
                        newProp(child, "base-type-builtin", base);
                }
        }
        if (sym->array_size) {
                /* TODO: modify get_expression_value to give error return */
                array_size = get_expression_value(sym->array_size);
                newNumProp(child, "array-size", array_size);
        }


        switch (sym->type) {
        case SYM_STRUCT:
        case SYM_UNION:
                examine_members(sym->symbol_list, child);
                break;
        case SYM_FN:
                examine_members(sym->arguments, child);
                break;
        case SYM_UNINITIALIZED:
                newProp(child, "base-type-builtin", builtin_typename(sym));
                break;
        default:
                break;
        }
        return;
}

static struct position *get_expansion_end (struct token *token)
{
        struct token *p1, *p2;

        for (p1=NULL, p2=NULL;
             !eof_token(token);
             p2 = p1, p1 = token, token = token->next);

        if (p2)
                return &(p2->pos);
        else
                return NULL;
}

static void examine_macro(struct symbol *sym, xmlNodePtr node)
{
        struct position *pos;

        /* this should probably go in the main codebase*/
        pos = get_expansion_end(sym->expansion);
        if (pos)
                sym->endpos = *pos;
        else
                sym->endpos = sym->pos;

        new_sym_node(sym, "macro", node);
}

static void examine_namespace(struct symbol *sym)
{
        if (sym->ident && sym->ident->reserved)
                return;

        switch(sym->namespace) {
        case NS_MACRO:
                examine_macro(sym, root_node);
```

```
259                 break;
260         case NS_TYPEDEF:
261         case NS_STRUCT:
262         case NS_SYMBOL:
263                 examine_symbol(sym, root_node);
264                 break;
265         case NS_NONE:
266         case NS_LABEL:
267         case NS_ITERATOR:
268         case NS_UNDEF:
269         case NS_PREPROCESSOR:
270         case NS_KEYWORD:
271                 break;
272         default:
273                 die("Unrecognised namespace type %d",sym->namespace);
274         }

276 }

278 static int get_stream_id (const char *name)
279 {
280         int i;
281         for (i=0; i<input_stream_nr; i++) {
282                 if (strcmp(name, stream_name(i))==0)
283                         return i;
284         }
285         return -1;
286 }

288 static inline void examine_symbol_list(const char *file, struct symbol_list *lis
289 {
290         struct symbol *sym;
291         int stream_id = get_stream_id (file);

293         if (!list)
294                 return;
295         FOR_EACH_PTR(list, sym) {
296                 if (sym->pos.stream == stream_id)
297                         examine_namespace(sym);
298         } END_FOR_EACH_PTR(sym);
299 }

301 int main(int argc, char **argv)
302 {
303         struct string_list *filelist = NULL;
304         struct symbol_list *symlist = NULL;
305         char *file;

307         doc = xmlNewDoc(BAD_CAST "1.0");
308         root_node = xmlNewNode(NULL, BAD_CAST "parse");
309         xmlDocSetRootElement(doc, root_node);

311 /* - A DTD is probably unnecessary for something like this

313         dtd = xmlCreateIntSubset(doc, "parse", "http://www.kernel.org/pub/softwa

315         ns = xmlNewNs (root_node, "http://www.kernel.org/pub/software/devel/spar

317         xmlSetNs(root_node, ns);
318 */
319         symlist = sparse_initialize(argc, argv, &filelist);

321         FOR_EACH_PTR_NOTAG(filelist, file) {
322                 examine_symbol_list(file, symlist);
323                 sparse_keep_tokens(file);
324                 examine_symbol_list(file, file_scope->symbols);
```



```
325                 examine_symbol_list(file, global_scope->symbols);
326         } END_FOR_EACH_PTR_NOTAG(file);


329         xmlSaveFormatFileEnc("-", doc, "UTF-8", 1);
330         xmlFreeDoc(doc);
331         xmlCleanupParser();

333         return 0;
334 }
```

```
**********************************************************
   11428 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/cgcc
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w
# ------------------------------------------------------------------------------

my $cc = $ENV{'REAL_CC'} || 'cc';
my $check = $ENV{'CHECK'} || 'sparse';
my $ccom = $cc;

my $m32 = 0;
my $m64 = 0;
my $has_specs = 0;
my $gendeps = 0;
my $do_check = 0;
my $do_compile = 1;
my $gcc_base_dir;
my $multiarch_dir;
my $verbose = 0;

while (@ARGV) {
    $_ = shift(@ARGV);
    # Look for a .c file.  We don't want to run the checker on .o or .so files
    # in the link run.  (This simplistic check knows nothing about options
    # with arguments, but it seems to do the job.)
    $do_check = 1 if /^[^-].*\.c$/;

    # Ditto for stdin.
    $do_check = 1 if $_ eq '-';

    $m32 = 1 if /^-m32$/;
    $m64 = 1 if /^-m64$/;
    $gendeps = 1 if /^-M$/;

    if (/^-target=(.*)$/) {
        $check .= &add_specs ($1);
        $has_specs = 1;
        next;
    }

    if ($_ eq '-no-compile') {
        $do_compile = 0;
        next;
    }

    if (/^-gcc-base-dir$/) {
        $gcc_base_dir = shift @ARGV;
        die ("$0: missing argument for -gcc-base-dir option") if !$gcc_base_dir;
        next;
    }

    if (/^-multiarch-dir$/) {
        $multiarch_dir = shift @ARGV;
        die ("$0: missing argument for -multiarch-dir option") if !$multiarch_di
        next;
    }

    # If someone adds "-E", don't pre-process twice.
    $do_compile = 0 if $_ eq '-E';

    $verbose = 1 if $_ eq '-v';

    my $this_arg = ' ' . &quote_arg ($_);
```



```
    $cc .= $this_arg unless &check_only_option ($_);
    $check .= $this_arg;
}

if ($gendeps) {
    $do_compile = 1;
    $do_check = 0;
}

if ($do_check) {
    if (!$has_specs) {
        $check .= &add_specs ('host_arch_specs');
        $check .= &add_specs ('host_os_specs');
    }

    $gcc_base_dir = qx($ccom -print-file-name=) if !$gcc_base_dir;
    chomp($gcc_base_dir);  # possibly remove '\n' from compiler
    $check .= " -gcc-base-dir " . $gcc_base_dir if $gcc_base_dir;

    $multiarch_dir = qx($ccom -print-multiarch) if ! defined $multiarch_dir;
    chomp($multiarch_dir);  # possibly remove '\n' from compiler
    $check .= " -multiarch-dir " . $multiarch_dir if $multiarch_dir;

    print "$check\n" if $verbose;
    if ($do_compile) {
        system ($check);
    } else {
        exec ($check);
    }
}

if ($do_compile) {
    print "$cc\n" if $verbose;
    exec ($cc);
}

exit 0;

# ------------------------------------------------------------------------------
# Check if an option is for "check" only.

sub check_only_option {
    my ($arg) = @_;
    return 1 if $arg =~ /^-W(no-?)?(address-space|bitwise|cast-to-as|cast-trunca
    return 1 if $arg =~ /^-v(no-?)?(entry|dead)$/;
    return 1 if $arg =~ /^-f(dump-linearize|memcpy-max-count)(=\S*)?$/;
    return 0;
}

# ------------------------------------------------------------------------------
# Simple arg-quoting function.  Just adds backslashes when needed.

sub quote_arg {
    my ($arg) = @_;
    return "''" if $arg eq '';
    return join ('',
                 map {
                     m|^[-a-zA-Z0-9._/,=]+$| ? $_ : "\\" . $_;
                 } (split (//, $arg)));
}

# ------------------------------------------------------------------------------

sub integer_types {
    my ($char,@dummy) = @_;
```

```
127     my %pow2m1 =
128         (8 => '127',
129          16 => '32767',
130          32 => '2147483647',
131          64 => '9223372036854775807',
132          128 => '170141183460469231731687303715884105727',
133          );
134     my @types = (['SCHAR',''], ['SHRT',''], ['INT',''], ['LONG','L'], ['LONG_LON

136     my $result = " -D__CHAR_BIT__=$char";
137     while (@types && @_) {
138         my $bits = shift @_;
139         my ($name,$suffix) = @{ shift @types };
140         die "$0: weird number of bits." unless exists $pow2m1{$bits};
141         $result .= " -D__${name}_MAX__=" . $pow2m1{$bits} . $suffix;
142     }
143     return $result;
144 }

146 # ------------------------------------------------------------------------

148 sub float_types {
149     my ($has_inf,$has_qnan,$dec_dig,@bitsizes) = @_;
150     my $result = " -D__FLT_RADIX__=2";
151     $result .= " -D__FINITE_MATH_ONLY__=" . ($has_inf || $has_qnan ? '0' : '1');
152     $result .= " -D__DECIMAL_DIG__=$dec_dig";

154     my %constants =
155         (24 =>
156          {
157              'MIN' => '1.17549435e-38',
158              'MAX' => '3.40282347e+38',
159              'EPSILON' => '1.19209290e-7',
160              'DENORM_MIN' => '1.40129846e-45',
161          },
162          53 =>
163          {
164              'MIN' => '2.2250738585072014e-308',
165              'MAX' => '1.7976931348623157e+308',
166              'EPSILON' => '2.2204460492503131e-16',
167              'DENORM_MIN' => '4.9406564584124654e-324',
168          },
169          64 =>
170          {
171              'MIN' => '3.3621031431120935062626778173217526e-4932',
172              'MAX' => '1.1897314953572317650857593266280070e+4932',
173              'EPSILON' => '1.0842021724855044340e-19',
174              'DENORM_MIN' => '3.6451995318824746025e-4951',
175          },
176          113 =>
177          {
178              'MIN' => '3.3621031431120935062626778173217526e-4932',
179              'MAX' => '1.1897314953572317650857593266280070e+4932',
180              'EPSILON' => '1.9259299443872358530559779425849273e-34',
181              'DENORM_MIN' => '6.4751751194380251109244389582276465e-4966',
182          },
183          );

185     my @types = (['FLT','F'], ['DBL',''], ['LDBL','L']);
186     while (@types) {
187         my ($mant_bits,$exp_bits) = @{ shift @bitsizes };
188         my ($name,$suffix) = @{ shift @types };

190         my $h = $constants{$mant_bits};
191         die "$0: weird number of mantissa bits." unless $h;
```



```
193         my $mant_dig = int (($mant_bits - 1) * log (2) / log (10));
194         my $max_exp = 1 << ($exp_bits - 1);
195         my $min_exp = 3 - $max_exp;
196         my $max_10_exp = int ($max_exp * log (2) / log (10));
197         my $min_10_exp = -int (-$min_exp * log (2) / log (10));

199         $result .= " -D__${name}_MANT_DIG__=$mant_bits";
200         $result .= " -D__${name}_DIG__=$mant_dig";
201         $result .= " -D__${name}_MIN_EXP__='($min_exp)'";
202         $result .= " -D__${name}_MAX_EXP__=$max_exp";
203         $result .= " -D__${name}_MIN_10_EXP__='($min_10_exp)'";
204         $result .= " -D__${name}_MAX_10_EXP__=$max_10_exp";
205         $result .= " -D__${name}_HAS_INFINITY__=" . ($has_inf ? '1' : '0');
206         $result .= " -D__${name}_HAS_QUIET_NAN__=" . ($has_qnan ? '1' : '0');;

208         foreach my $inf (sort keys %$h) {
209             $result .= " -D__${name}_${inf}__=" . $h->{$inf} . $suffix;
210         }
211     }
212     return $result;
213 }

215 # ------------------------------------------------------------------------

217 sub define_size_t {
218     my ($text) = @_;
219     # We have to undef in order to override check's internal definition.
220     return ' -U__SIZE_TYPE__ ' . &quote_arg ("-D__SIZE_TYPE__=$text");
221 }

223 # ------------------------------------------------------------------------

225 sub add_specs {
226     my ($spec) = @_;
227     if ($spec eq 'sunos') {
228         return &add_specs ('unix') .
229             ' -D__sun__=1 -D__sun=1 -Dsun=1' .
230             ' -D__svr4__=1 -DSVR4=1' .
231             ' -D__STDC__=0' .
232             ' -D_REENTRANT' .
233             ' -D_SOLARIS_THREADS' .
234             ' -DNULL="((void *)0)"';
235     } elsif ($spec eq 'linux') {
236         return &add_specs ('unix') .
237             ' -D__linux__=1 -D__linux=1 -Dlinux=linux';
238     } elsif ($spec eq 'gnu/kfreebsd') {
239         return &add_specs ('unix') .
240             ' -D__FreeBSD_kernel__=1';
241     } elsif ($spec eq 'openbsd') {
242         return &add_specs ('unix') .
243             ' -D__OpenBSD__=1';
244     } elsif ($spec eq 'darwin') {
245         return
246             ' -D__APPLE__=1 -D__MACH__=1';
247     } elsif ($spec eq 'unix') {
248         return ' -Dunix=1 -D__unix=1 -D__unix__=1';
249     } elsif ( $spec =~ /^cygwin/) {
250         return &add_specs ('unix') .
251             ' -D__CYGWIN__=1 -D__CYGWIN32__=1' .
252             " -D'__cdecl=__attribute__((__cdecl__))'" .
253             " -D'__cdecl=__attribute__((__cdecl__))'" .
254             " -D'__stdcall=__attribute__((__stdcall__))'" .
255             " -D'__stdcall=__attribute__((__stdcall__))'" .
256             " -D'__fastcall=__attribute__((__fastcall__))'" .
257             " -D'__fastcall=__attribute__((__fastcall__))'" .
258             " -D'__declspec(x)=__attribute__((x))'";
```

```
259     } elsif ($spec eq 'i86') {
260         return (' -D__i386=1 -D__i386__=1' .
261                 &integer_types (8, 16, 32, $m64 ? 64 : 32, 64) .
262                 &float_types (1, 1, 21, [24,8], [53,11], [64,15]) .
263                 &define_size_t ($m64 ? "long unsigned int" : "unsigned int") .
264                 ' -D__SIZEOF_POINTER__=' . ($m64 ? '8' : '4'));
265     } elsif ($spec eq 'sparc') {
266         return (' -D__sparc=1 -D__sparc__=1' .
267                 &integer_types (8, 16, 32, $m64 ? 64 : 32, 64) .
268                 &float_types (1, 1, 33, [24,8], [53,11], [113,15]) .
269                 &define_size_t ($m64 ? "long unsigned int" : "unsigned int") .
270                 ' -D__SIZEOF_POINTER__=' . ($m64 ? '8' : '4'));
271     } elsif ($spec eq 'sparc64') {
272         return (' -D__sparc=1 -D__sparc__=1 -D__sparcv9__=1 -D__sparc64__=1 -D__
273                 &integer_types (8, 16, 32, 64, 64, 128) .
274                 &float_types (1, 1, 33, [24,8], [53,11], [113,15]) .
275                 &define_size_t ("long unsigned int") .
276                 ' -D__SIZEOF_POINTER__=8');
277     } elsif ($spec eq 'x86_64') {
278         return (' -D__x86_64=1 -D__x86_64__=1' . ($m32 ? '' : ' -D__LP64__=1') .
279                 &integer_types (8, 16, 32, $m32 ? 32 : 64, 64, 128) .
280                 &float_types (1, 1, 33, [24,8], [53,11], [113,15]) .
281                 &define_size_t ($m32 ? "unsigned int" : "long unsigned int") .
282                 ' -D__SIZEOF_POINTER__=' . ($m32 ? '4' : '8'));
283     } elsif ($spec eq 'ppc') {
284         return (' -D__powerpc__=1 -D_BIG_ENDIAN -D_STRING_ARCH_unaligned=1' .
285                 &integer_types (8, 16, 32, $m64 ? 64 : 32, 64) .
286                 &float_types (1, 1, 21, [24,8], [53,11], [113,15]) .
287                 &define_size_t ($m64 ? "long unsigned int" : "unsigned int") .
288                 ' -D__SIZEOF_POINTER__=' . ($m64 ? '8' : '4'));
289     } elsif ($spec eq 'ppc64') {
290         return (' -D__powerpc__=1 -D__PPC__=1 -D_STRING_ARCH_unaligned=1' .
291                 ' -D__powerpc64__=1 -D__PPC64__=1' .
292                 ' -m64' .
293                 &float_types (1, 1, 21, [24,8], [53,11], [113,15]));
294     } elsif ($spec eq 's390x') {
295         return (' -D__s390x__ -D__s390__ -D_BIG_ENDIAN' .
296                 &integer_types (8, 16, 32, $m64 ? 64 : 32, 64) .
297                 &float_types (1, 1, 36, [24,8], [53,11], [113,15]) .
298                 &define_size_t ("long unsigned int") .
299                 ' -D__SIZEOF_POINTER__=' . ($m64 ? '8' : '4'));
300     } elsif ($spec eq 'arm') {
301         chomp (my $gccmachine = `$cc -dumpmachine`);
302         my $cppsymbols = ' -D__arm__=1 -m32';

304         if ($gccmachine eq 'arm-linux-gnueabihf') {
305             $cppsymbols .= ' -D__ARM_PCS_VFP=1';
306         }

308         return ($cppsymbols .
309                 &float_types (1, 1, 36, [24,8], [53,11], [53, 11]));
310     } elsif ($spec eq 'aarch64') {
311         return (' -D__aarch64__=1 -m64' .
312                 &float_types (1, 1, 36, [24,8], [53,11], [113,15]));
313     } elsif ($spec eq 'host_os_specs') {
314         my $os = `uname -s`;
315         chomp $os;
316         return &add_specs (lc $os);
317     } elsif ($spec eq 'host_arch_specs') {
318         my $arch = `uname -m`;
319         chomp $arch;
320         if ($arch =~ /^(i.?86|athlon)$/i) {
321             return &add_specs ('i86');
322         } elsif ($arch =~ /^(sun4u)$/i) {
323             return &add_specs ('sparc');
324         } elsif ($arch =~ /^(x86_64)$/i) {
```



```
325             return &add_specs ('x86_64');
326         } elsif ($arch =~ /^(ppc)$/i) {
327             return &add_specs ('ppc');
328         } elsif ($arch =~ /^(ppc64)$/i) {
329             return &add_specs ('ppc64') . ' -mbig-endian -D_CALL_ELF=1';
330         } elsif ($arch =~ /^(ppc64le)$/i) {
331             return &add_specs ('ppc64') . ' -mlittle-endian -D_CALL_ELF=2';
332         } elsif ($arch =~ /^(s390x)$/i) {
333             return &add_specs ('s390x');
334         } elsif ($arch =~ /^(sparc64)$/i) {
335             return &add_specs ('sparc64');
336         } elsif ($arch =~ /^arm(?:v[78]l)?$/i) {
337             return &add_specs ('arm');
338         } elsif ($arch =~ /^(aarch64)$/i) {
339             return &add_specs ('aarch64');
340         }
341     } else {
342         die "$0: invalid specs: $spec\n";
343     }
344 }

346 # ----------------------------------------------------------------------------
```

```
**********************************************************
    1062 Fri Dec 21 15:00:00 2018
new/usr/src/tools/smatch/src/cgcc.1
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
.\" cgcc manpage by Josh Triplett
.TH cgcc "1"
.
.SH NAME
cgcc \- Compiler wrapper to run Sparse after compiling
.
.SH SYNOPSIS
.B cgcc
[\fISPARSE OPTIONS\fR]... [\fICOMPILER OPTIONS\fR]... [\fIINPUT FILES\fR]...
.br
.B make CC=cgcc
.
.SH DESCRIPTION
\fBcgcc\fR provides a wrapper around a C compiler (\fBcc\fR by
default) which also invokes the Sparse static analysis tool.
.P
\fBcgcc\fR accepts all Sparse command-line options, such as warning
options, and passes all other options through to the compiler.
.P
By providing the same interface as the C compiler, \fBcgcc\fR allows
projects to run Sparse as part of their build without modifying their
build system, by using \fBcgcc\fR as the compiler.  For many projects,
setting \fBCC=cgcc\fR on the \fBmake\fR command-line will work.
.
.SH ENVIRONMENT
.TP
.B REAL_CC
If set, \fBcgcc\fR will use this as the compiler to invoke, rather
than the default \fBcc\fR.
.
.TP
.B CHECK
If set, \fBcgcc\fR will use this as the Sparse program to invoke,
rather than the default \fBsparse\fR.
.
.SH SEE ALSO
.BR sparse (1)
```

```
**********************************************************
    3310 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/char.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <string.h>
#include "target.h"
#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "expression.h"
#include "char.h"

static const char *parse_escape(const char *p, unsigned *val, const char *end, i
{
        unsigned c = *p++;
        unsigned d;
        if (c != '\\') {
                *val = c;
                return p;
        }

        c = *p++;
        switch (c) {
        case 'a': c = '\a'; break;
        case 'b': c = '\b'; break;
        case 't': c = '\t'; break;
        case 'n': c = '\n'; break;
        case 'v': c = '\v'; break;
        case 'f': c = '\f'; break;
        case 'r': c = '\r'; break;
        case 'e': c = '\e'; break;
        case 'x': {
                unsigned mask = -(1U << (bits - 4));
                for (c = 0; p < end; c = (c << 4) + d) {
                        d = hexval(*p);
                        if (d > 16)
                                break;
                        p++;
                        if (c & mask) {
                                warning(pos,
                                        "hex escape sequence out of range");
                                mask = 0;
                        }
                }
                break;
        }
        case '0'...'7': {
                if (p + 2 < end)
                        end = p + 2;
                c -= '0';
                while (p < end && (d = *p - '0') < 8) {
                        c = (c << 3) + d;
                        p++;
                }
                if ((c & 0400) && bits < 9)
                        warning(pos,
                                "octal escape sequence out of range");
                break;
        }
        default:        /* everything else is left as is */
                warning(pos, "unknown escape sequence: '\\%c'", c);
                break;
        case '\\':
        case '\'':
```



```
        case '"':
        case '?':
                break;  /* those are legal, so no warnings */
        }
        *val = c & ~((~0U << (bits - 1)) << 1);
        return p;
}

void get_char_constant(struct token *token, unsigned long long *val)
{
        const char *p = token->embedded, *end;
        unsigned v;
        int type = token_type(token);
        switch (type) {
        case TOKEN_CHAR:
        case TOKEN_WIDE_CHAR:
                p = token->string->data;
                end = p + token->string->length - 1;
                break;
        case TOKEN_CHAR_EMBEDDED_0 ... TOKEN_CHAR_EMBEDDED_3:
                end = p + type - TOKEN_CHAR;
                break;
        default:
                end = p + type - TOKEN_WIDE_CHAR;
        }
        p = parse_escape(p, &v, end,
                        type < TOKEN_WIDE_CHAR ? bits_in_char : bits_in_wchar, t
        if (p != end)
                warning(token->pos,
                        "multi-character character constant");
        *val = v;
}

struct token *get_string_constant(struct token *token, struct expression *expr)
{
        struct string *string = token->string;
        struct token *next = token->next, *done = NULL;
        int stringtype = token_type(token);
        int is_wide = stringtype == TOKEN_WIDE_STRING;
        static char buffer[MAX_STRING];
        int len = 0;
        int bits;
        int esc_count = 0;

        while (!done) {
                switch (token_type(next)) {
                case TOKEN_WIDE_STRING:
                        is_wide = 1;
                case TOKEN_STRING:
                        next = next->next;
                        break;
                default:
                        done = next;
                }
        }
        bits = is_wide ? bits_in_wchar : bits_in_char;
        while (token != done) {
                unsigned v;
                const char *p = token->string->data;
                const char *end = p + token->string->length - 1;
                while (p < end) {
                        if (*p == '\\')
                                esc_count++;
                        p = parse_escape(p, &v, end, bits, token->pos);
                        if (len < MAX_STRING)
                                buffer[len] = v;
```

```
127                         len++;
128                 }
129                 token = token->next;
130         }
131         if (len > MAX_STRING) {
132                 warning(token->pos, "trying to concatenate %d-character string (
133                 len = MAX_STRING;
134         }

136         if (esc_count || len >= string->length) {
137                 if (string->immutable || len >= string->length) /* can't canniba
138                         string = __alloc_string(len+1);
139                 string->length = len+1;
140                 memcpy(string->data, buffer, len);
141                 string->data[len] = '\0';
142         }
143         expr->string = string;
144         expr->wide = is_wide;
145         return token;
146 }
```

```
**********************************************************
     148 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/char.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 extern void get_char_constant(struct token *, unsigned long long *);
   2 extern struct token *get_string_constant(struct token *, struct expression *);
```

```
**********************************************************
    1739 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_64bit_shift.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_shift_assignment(struct expression *expr)
{
        struct symbol *left_type, *right_type;
        struct expression *right;
        sval_t sval;
        sval_t bits, shifter;
        char *name;

        right = strip_expr(expr->right);
        if (right->type != EXPR_BINOP || right->op != SPECIAL_LEFTSHIFT)
                return;

        left_type = get_type(expr->left);
        if (left_type != &llong_ctype && left_type != &ullong_ctype)
                return;

        right_type = get_type(expr->right);

        if (type_bits(right_type) == 64)
                return;

        if (get_value(right, &sval))
                return;

        get_absolute_max(right->left, &bits);
        get_absolute_max(right->right, &shifter);

        bits = sval_cast(&ullong_ctype, bits);
        if (sval_cmp_val(shifter, 32) < 0) {
                sval = sval_binop(bits, SPECIAL_LEFTSHIFT, shifter);
                if (sval_cmp_val(sval, UINT_MAX) < 0)
                        return;
        }

        name = expr_to_str_sym(right, NULL);
        sm_warning("should '%s' be a 64 bit type?", name);
        free_string(name);
}
```



```
void check_64bit_shift(int id)
{
        my_id = id;

        add_hook(&match_shift_assignment, ASSIGNMENT_HOOK);
}
```

```
**********************************************************
    2858 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_access_ok_math.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static int can_overflow(struct expression *expr)
{
        sval_t max;
        int uncapped = 0;

        expr = strip_expr(expr);

        if (expr->type == EXPR_BINOP) {
                uncapped += can_overflow(expr->left);
                uncapped += can_overflow(expr->right);

                if (uncapped &&
                        (expr->op == '+' || expr->op == '*' || expr->op == SPECI
                        return 1;

                return 0;
        }

        if (get_implied_max(expr, &max))
                return 0;
        if (get_absolute_max(expr, &max) && sval_cmp_val(max, 4096) <= 0)
                return 0;
        return 1;
}

static void match_size(struct expression *size_expr)
{
        char *name;

        size_expr = strip_expr(size_expr);
        if (!size_expr)
                return;
        if (size_expr->type != EXPR_BINOP) {
                size_expr = get_assigned_expr(size_expr);
                if (!size_expr || size_expr->type != EXPR_BINOP)
                        return;
        }
        if (!can_overflow(size_expr))
                return;
```



```
        name = expr_to_str(size_expr);
        sm_warning("math in access_ok() is dangerous '%s'", name);

        free_string(name);
}

static void match_access_ok(const char *fn, struct expression *expr, void *data)
{
        struct expression *size_expr;

        size_expr = get_argument_from_call_expr(expr->args, 1);
        match_size(size_expr);
}

static void split_asm_constraints(struct expression_list *expr_list)
{
        struct expression *expr;
        int state = 0;
        int i;

        i = 0;
        FOR_EACH_PTR(expr_list, expr) {

                switch (state) {
                case 0: /* identifier */
                case 1: /* constraint */
                        state++;
                        continue;
                case 2: /* expression */
                        state = 0;
                        if (i == 1)
                                match_size(expr);
                        i++;
                        continue;
                }
        } END_FOR_EACH_PTR(expr);
}

static void match_asm_stmt(struct statement *stmt)
{
        char *name;

        name = get_macro_name(stmt->pos);
        if (!name || strcmp(name, "access_ok") != 0)
                return;
        split_asm_constraints(stmt->asm_inputs);
}

void check_access_ok_math(int id)
{
        my_id = id;
        if (option_project != PROJ_KERNEL)
                return;
        if (!option_spammy)
                return;
        add_function_hook("__access_ok", &match_access_ok, NULL);
        add_hook(&match_asm_stmt, ASM_HOOK);
}
```

```
**********************************************************
    2562 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_all_func_returns.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright 2018 Joyent, Inc.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Like lint of old, check that every return value from every function is used.
 * Casting to (void) will silence this check.
 */

#include "smatch.h"
#include "smatch_slist.h"

static void check_func_return(struct expression *expr)
{
        struct symbol *sym = get_real_base_type(get_type(expr->fn));
        const char *func = expr_to_str(expr->fn);
        struct statement *stmt;

        if (sym == NULL) {
                sm_error("unknown type for func '%s'", func);
                return;
        }

        if (expr->type != EXPR_CALL) {
                sm_error("func '%s' is not a call site", func);
                return;
        }

        /*
         * There is never any need to check these returns.
         */
        if (strcmp(func, "memcpy") == 0 ||
            strcmp(func, "memmove") == 0 ||
            strcmp(func, "memset") == 0)
                return;

        /*
         * Closer to a policy here, but there seems very few cases where it's
         * useful to check the return value of the standard printf() family
         * outputting to stdout or stderr.
         */
        if (strcmp(func, "printf") == 0 || strcmp(func, "vprintf") == 0)
                return;

        if (strcmp(func, "fprintf") == 0 || strcmp(func, "vfprintf")) {
                const char *arg0 = expr_to_str(get_argument_from_call_expr(expr-
```



```
                if (arg0 != NULL &&
                    (strcmp(arg0, "(&__iob[1])") == 0 ||
                    strcmp(arg0, "(&__iob[2])") == 0))
                        return;
        }

        /*
         * Either we got the return type already (direct call),
         * or we need to go one further (function pointer call)
         */
        if (sym == &void_ctype || (sym->type == SYM_FN &&
                get_real_base_type(sym) == &void_ctype))
                return;

        stmt = last_ptr_list((struct ptr_list *)big_statement_stack);

        if (stmt->type == STMT_EXPRESSION && stmt->expression == expr)
                sm_error("unchecked function return '%s'", expr_to_str(expr->fn)
}

void check_all_func_returns(int id)
{
        if (option_project != PROJ_ILLUMOS_KERNEL &&
            option_project != PROJ_ILLUMOS_USER)
                return;

        add_hook(&check_func_return, FUNCTION_CALL_HOOK);
}
```

**********
*1514 Fri Dec 21 15:00:01 2018*
*new/usr/src/tools/smatch/src/check_allocating_enough_data.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static void db_returns_buf_size(struct expression *expr, int param, char *unused
{
        struct expression *call;
        struct symbol *left_type, *right_type;
        int bytes;
        sval_t sval;

        if (expr->type != EXPR_ASSIGNMENT)
                return;
        right_type = get_pointer_type(expr->right);
        if (!right_type || type_bits(right_type) != -1)
                return;

        call = strip_expr(expr->right);
        left_type = get_pointer_type(expr->left);

        if (!parse_call_math(call, math, &sval) || sval.value == 0)
                return;
        if (!left_type)
                return;
        bytes = type_bytes(left_type);
        if (bytes <= 0)
                return;
        if (sval.uvalue >= bytes)
                return;
        sm_error("not allocating enough data %d vs %s", bytes, sval_to_str(sval)
}

void check_allocating_enough_data(int id)
{
        select_return_states_hook(BUF_SIZE, &db_returns_buf_size);
}
```

```
**********************************************************
    2530 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_allocation_funcs.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <fcntl.h>
#include <unistd.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

/*
 * Print a list of functions that return newly allocated memory.
 */

static struct tracker_list *allocated;

static const char *allocation_funcs[] = {
        "kmalloc",
        "kzalloc",
        "kcalloc",
        "__alloc_skb",
        NULL,
};

static void match_allocation(const char *fn, struct expression *expr,
                             void *info)
{
        char *left_name;
        struct symbol *left_sym;

        left_name = expr_to_var_sym(expr->left, &left_sym);
        if (!left_name || !left_sym)
                goto free;
        if (left_sym->ctype.modifiers &
            (MOD_NONLOCAL | MOD_STATIC | MOD_ADDRESSABLE))
                goto free;
        add_tracker(&allocated, my_id, left_name, left_sym);
free:
        free_string(left_name);
}

static int returns_new_stuff = 0;
static int returns_old_stuff = 0;
static void match_return(struct expression *ret_value)
{
```



```
        char *name;
        struct symbol *sym;
        sval_t tmp;

        if (__inline_fn)
                return;
        if (get_value(ret_value, &tmp) && tmp.value == 0)
                return;
        returns_new_stuff = 1;
        name = expr_to_var_sym(ret_value, &sym);
        if (!name || !sym) {
                returns_old_stuff = 1;
                goto free;
        }
        if (!in_tracker_list(allocated, my_id, name, sym))
                returns_old_stuff = 1;
free:
        free_string(name);
}

static void match_end_func(struct symbol *sym)
{
        if (__inline_fn)
                return;
        if (returns_new_stuff && !returns_old_stuff)
                sm_info("allocation func");
        free_trackers_and_list(&allocated);
        returns_new_stuff = 0;
        returns_old_stuff = 0;
}

void check_allocation_funcs(int id)
{
        int i;

        if (!option_info || option_project != PROJ_KERNEL)
                return;

        my_id = id;
        add_hook(&match_return, RETURN_HOOK);
        add_hook(&match_end_func, AFTER_FUNC_HOOK);
        for (i = 0; allocation_funcs[i]; i++) {
                add_function_assign_hook(allocation_funcs[i],
                                         &match_allocation, NULL);
        }
}
```

```
**********************************************************
    1281 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_array_condition.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * struct foo { char buf[10]; };
 *
 * struct foo *p = something();
 * if (p->buf) { ...
 *
 */

#include "smatch.h"

static int my_id;

static void match_condition(struct expression *expr)
{
        struct symbol *type;
        char *str;

        if (expr->type != EXPR_DEREF)
                return;
        type = get_type(expr);
        if (!type || type->type != SYM_ARRAY)
                return;
        if (get_macro_name(expr->pos))
                return;

        str = expr_to_str(expr);
        sm_warning("this array is probably non-NULL. '%s'", str);
        free_string(str);
}

void check_array_condition(int id)
{
        my_id = id;
        add_hook(&match_condition, CONDITION_HOOK);
}
```

```
**********************************************************
    1563 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_assign_vs_compare.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void check_constant(struct expression *expr)
{
        sval_t val;

        if (!get_value(expr->right, &val))
                return;
        sm_warning("was '== %s' instead of '='", sval_to_str(val));
}

static void check_address(struct expression *expr)
{
        char *str;
        struct expression *right = strip_expr(expr->right);

        if (!__cur_stmt || __cur_stmt->type != STMT_IF)
                return;

        if (right->type != EXPR_PREOP ||
            right->op != '&')
                return;

        if (get_macro_name(expr->pos))
                return;

        str = expr_to_str(right);
        sm_warning("was '== %s' instead of '='", str);
        free_string(str);
}

static void match_condition(struct expression *expr)
{
        if (expr->type != EXPR_ASSIGNMENT || expr->op != '=')
                return;

        check_constant(expr);
        check_address(expr);
}

void check_assign_vs_compare(int id)
```



```
{
        my_id = id;
        add_hook(&match_condition, CONDITION_HOOK);
}
```

```
**********************************************************
    6660 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_atomic_inc_dec.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2016 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

STATE(inc);
STATE(orig);
STATE(dec);

static void db_inc_dec(struct expression *expr, int param, const char *key, cons
{
        struct smatch_state *start_state;
        struct expression *arg;
        char *name;
        struct symbol *sym;

        while (expr->type == EXPR_ASSIGNMENT)
                expr = strip_expr(expr->right);
        if (expr->type != EXPR_CALL)
                return;

        arg = get_argument_from_call_expr(expr->args, param);
        if (!arg)
                return;

        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        start_state = get_state(my_id, name, sym);

        if (inc_dec == ATOMIC_INC) {
//              if (start_state == &inc)
//                      sm_error("XXX double increment '%s'", name);
                set_state(my_id, name, sym, &inc);
        } else {
//              if (start_state == &dec)
//                      sm_error("XXX double decrement '%s'", name);
                if (start_state == &inc)
                        set_state(my_id, name, sym, &orig);
                else
                        set_state(my_id, name, sym, &dec);
        }
free:
        free_string(name);
}

static void db_inc(struct expression *expr, int param, char *key, char *value)
{
        db_inc_dec(expr, param, key, value, ATOMIC_INC);
}

static void db_dec(struct expression *expr, int param, char *key, char *value)
{
        db_inc_dec(expr, param, key, value, ATOMIC_DEC);
}

static void match_atomic_inc(const char *fn, struct expression *expr, void *_unu
{
        db_inc_dec(expr, 0, "$->counter", "", ATOMIC_INC);
}

static void match_atomic_dec(const char *fn, struct expression *expr, void *_unu
{
        db_inc_dec(expr, 0, "$->counter", "", ATOMIC_DEC);
}

static void match_atomic_add(const char *fn, struct expression *expr, void *_unu
{
        struct expression *amount;
        sval_t sval;

        amount = get_argument_from_call_expr(expr->args, 0);
        if (get_implied_value(amount, &sval) && sval_is_negative(sval)) {
                db_inc_dec(expr, 1, "$->counter", "", ATOMIC_DEC);
                return;
        }

        db_inc_dec(expr, 1, "$->counter", "", ATOMIC_INC);
}

static void match_atomic_sub(const char *fn, struct expression *expr, void *_unu
{
        db_inc_dec(expr, 1, "$->counter", "", ATOMIC_DEC);
}

static void refcount_inc(const char *fn, struct expression *expr, void *param)
{
        db_inc_dec(expr, PTR_INT(param), "$->ref.counter", "", ATOMIC_INC);
}

static void refcount_dec(const char *fn, struct expression *expr, void *param)
{
        db_inc_dec(expr, PTR_INT(param), "$->ref.counter", "", ATOMIC_DEC);
}

static void match_return_info(int return_id, char *return_ranges, struct express
{
        struct sm_state *sm;
        const char *param_name;
        int param;

        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
                if (sm->state != &inc &&
                    sm->state != &dec)
                        continue;
                param = get_param_num_from_sym(sm->sym);
```

```
127                 if (param < 0)
128                         continue;
129                 param_name = get_param_name(sm);
130                 if (!param_name)
131                         continue;
132                 sql_insert_return_states(return_id, return_ranges,
133                                          (sm->state == &inc) ? ATOMIC_INC : ATOM
134                                          param, param_name, "");
135         } END_FOR_EACH_SM(sm);
136 }
137
138 enum {
139         NEGATIVE, ZERO, POSITIVE,
140 };
141
142 static int success_fail_positive(struct range_list *rl)
143 {
144         if (!rl)
145                 return ZERO;
146
147         if (sval_is_negative(rl_min(rl)))
148                 return NEGATIVE;
149
150         if (rl_min(rl).value == 0)
151                 return ZERO;
152
153         return POSITIVE;
154 }
155
156 static void check_counter(const char *name, struct symbol *sym)
157 {
158         struct range_list *inc_lines = NULL;
159         struct range_list *dec_lines = NULL;
160         int inc_buckets[3] = {};
161         struct stree *stree;
162         struct sm_state *return_sm;
163         struct sm_state *sm;
164         sval_t line = sval_type_val(&int_ctype, 0);
165
166         FOR_EACH_PTR(get_all_return_strees(), stree) {
167                 return_sm = get_sm_state_stree(stree, RETURN_ID, "return_ranges"
168                 if (!return_sm)
169                         continue;
170                 line.value = return_sm->line;
171
172                 sm = get_sm_state_stree(stree, my_id, name, sym);
173                 if (!sm)
174                         continue;
175
176                 if (sm->state != &inc && sm->state != &dec)
177                         continue;
178
179                 if (sm->state == &inc) {
180                         add_range(&inc_lines, line, line);
181                         inc_buckets[success_fail_positive(estate_rl(return_sm->s
182                 }
183                 if (sm->state == &dec)
184                         add_range(&dec_lines, line, line);
185         } END_FOR_EACH_PTR(stree);
186
187         if (inc_buckets[NEGATIVE] &&
188             inc_buckets[ZERO]) {
189                 // sm_warning("XXX '%s' not decremented on lines: %s.", name, sh
190         }
191
192 }
```



```
193
194 static void match_check_missed(struct symbol *sym)
195 {
196         struct sm_state *sm;
197
198         FOR_EACH_MY_SM(my_id, get_all_return_states(), sm) {
199                 check_counter(sm->name, sm->sym);
200         } END_FOR_EACH_SM(sm);
201 }
202
203 int on_atomic_dec_path(void)
204 {
205         struct sm_state *sm;
206
207         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
208                 if (sm->state == &dec)
209                         return 1;
210         } END_FOR_EACH_SM(sm);
211
212         return 0;
213 }
214
215 int was_inced(const char *name, struct symbol *sym)
216 {
217         return get_state(my_id, name, sym) == &inc;
218 }
219
220 void check_atomic_inc_dec(int id)
221 {
222         my_id = id;
223
224         if (option_project != PROJ_KERNEL)
225                 return;
226
227         select_return_states_hook(ATOMIC_INC, &db_inc);
228         select_return_states_hook(ATOMIC_DEC, &db_dec);
229         add_function_hook("atomic_inc_return", &match_atomic_inc, NULL);
230         add_function_hook("atomic_add_return", &match_atomic_add, NULL);
231         add_function_hook("atomic_sub_return", &match_atomic_sub, NULL);
232         add_function_hook("atomic_sub_and_test", &match_atomic_sub, NULL);
233         add_function_hook("atomic_dec_and_test", &match_atomic_dec, NULL);
234         add_function_hook("_atomic_dec_and_lock", &match_atomic_dec, NULL);
235         add_function_hook("atomic_dec", &match_atomic_dec, NULL);
236         add_function_hook("atomic_long_inc", &match_atomic_inc, NULL);
237         add_function_hook("atomic_long_dec", &match_atomic_dec, NULL);
238         add_function_hook("atomic_inc", &match_atomic_inc, NULL);
239         add_function_hook("atomic_sub", &match_atomic_sub, NULL);
240         add_split_return_callback(match_return_info);
241
242         add_function_hook("refcount_add_not_zero", &refcount_inc, INT_PTR(1));
243         add_function_hook("refcount_inc_not_zero", &refcount_inc, INT_PTR(0));
244         add_function_hook("refcount_sub_and_test", &refcount_dec, INT_PTR(1));
245         add_function_hook("refcount_dec_and_test", &refcount_dec, INT_PTR(1));
246
247         add_hook(&match_check_missed, END_FUNC_HOOK);
248 }
```

```
**********************************************************
    4097 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_bit_shift.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This test is used to warn about mixups between bit shifters and bit flags.
 *
 */

#include "smatch.h"
#include "smatch_function_hashtable.h"

static int my_id;

static DEFINE_HASHTABLE_INSERT(insert_struct, char, int);
static DEFINE_HASHTABLE_SEARCH(search_struct, char, int);
static struct hashtable *shifters;

static const char *get_shifter(struct expression *expr)
{
        const char *name;
        sval_t expr_value;
        const int *shifter_value;

        expr = strip_expr(expr);
        if (expr->type != EXPR_VALUE)
                return NULL;
        if (!get_value(expr, &expr_value))
                return NULL;
        name = pos_ident(expr->pos);
        if (!name)
                return NULL;
        shifter_value = search_struct(shifters, (char *)name);
        if (!shifter_value)
                return NULL;
        if (sval_cmp_val(expr_value, *shifter_value) != 0)
                return NULL;
        return name;
}

static void match_assign(struct expression *expr)
{
        const char *name;

        if (expr->op != SPECIAL_OR_ASSIGN)
                return;
        if (positions_eq(expr->pos, expr->right->pos))
```



```
                return;
        name = get_shifter(expr->right);
        if (!name)
                return;

        sm_warning("'%s' is a shifter (not for '%s').",
                        name, show_special(expr->op));
}

static void match_binop(struct expression *expr)
{
        const char *name;

        if (positions_eq(expr->pos, expr->right->pos))
                return;
        if (expr->op != '&')
                return;
        name = get_shifter(expr->right);
        if (!name)
                return;

        sm_warning("bit shifter '%s' used for logical '%s'",
                        name, show_special(expr->op));
}

static void register_shifters(void)
{
        char filename[256];
        struct token *token;
        char *name;
        int *val;

        snprintf(filename, sizeof(filename), "%s.bit_shifters", option_project_s
        token = get_tokens_file(filename);
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                name = alloc_string(show_ident(token->ident));
                token = token->next;
                if (token_type(token) != TOKEN_NUMBER)
                        return;
                val = malloc(sizeof(int));
                *val = atoi(token->number);
                insert_struct(shifters, name, val);
                token = token->next;
        }
        clear_token_alloc();
}

static void match_binop_info(struct expression *expr)
{
        char *name;
        sval_t sval;

        if (positions_eq(expr->pos, expr->right->pos))
                return;
        if (expr->op != SPECIAL_LEFTSHIFT)
                return;
        if (expr->right->type != EXPR_VALUE)
                return;
        name = pos_ident(expr->right->pos);
```

```
127         if (!name)
128                 return;
129         if (!get_value(expr->right, &sval))
130                 return;
131         sm_msg("info: bit shifter '%s' '%s'", name, sval_to_str(sval));
132 }

134 static void match_call(const char *fn, struct expression *expr, void *_arg_no)
135 {
136         struct expression *arg_expr;
137         int arg_no = PTR_INT(_arg_no);
138         sval_t sval;
139         char *name;

141         arg_expr = get_argument_from_call_expr(expr->args, arg_no);
142         if (positions_eq(expr->pos, arg_expr->pos))
143                 return;
144         name = pos_ident(arg_expr->pos);
145         if (!name)
146                 return;
147         if (!get_value(arg_expr, &sval))
148                 return;
149         sm_msg("info: bit shifter '%s' '%s'", name, sval_to_str(sval));
150 }

152 void check_bit_shift(int id)
153 {
154         my_id = id;

156         shifters = create_function_hashtable(5000);
157         register_shifters();

159         add_hook(&match_assign, ASSIGNMENT_HOOK);
160         add_hook(&match_binop, BINOP_HOOK);

162         if (option_info) {
163                 add_hook(&match_binop_info, BINOP_HOOK);
164                 if (option_project == PROJ_KERNEL) {
165                         add_function_hook("set_bit", &match_call, INT_PTR(0));
166                         add_function_hook("test_bit", &match_call, INT_PTR(0));
167                 }
168         }
169 }
```

**********
*1189 Fri Dec 21 15:00:01 2018*
*new/usr/src/tools/smatch/src/check_bogus_irqrestore.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Copyright (C) 2011 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_irqrestore(const char *fn, struct expression *expr, void *data
{
        struct expression *arg_expr;
        sval_t tmp;

        arg_expr = get_argument_from_call_expr(expr->args, 1);
        if (!get_implied_value(arg_expr, &tmp))
                return;
        sm_error("calling '%s()' with bogus flags", fn);
}

void check_bogus_irqrestore(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        my_id = id;
        add_function_hook("spin_unlock_irqrestore", &match_irqrestore, NULL);
}
```

```
**********************************************************
    2285 Fri Dec 21 15:00:01 2018
new/usr/src/tools/smatch/src/check_bogus_loop.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static int right_side_changes(struct expression *expr)
{
        sval_t dummy;

        if (get_value(expr->right, &dummy))
                return 0;
        return 1;
}

static struct expression *get_iterator_set(struct statement *stmt)
{
        struct expression *expr;

        if (!stmt)
                return NULL;
        if (stmt->type != STMT_EXPRESSION)
                return NULL;
        expr = stmt->expression;
        if (expr->type != EXPR_ASSIGNMENT)
                return NULL;
        if (expr->op != '=')
                return NULL;
        if (right_side_changes(expr))
                return NULL;
        return expr->left;
}

static struct expression *get_iterator_tested(struct expression *expr)
{
        if (!expr)
                return NULL;
        if (expr->type != EXPR_COMPARE)
                return NULL;
        return expr->left;
}

static void match_loop(struct statement *stmt)
{
```



```
        struct expression *iterator;
        char *iter_set;
        char *iter_tested;

        if (get_macro_name(stmt->pos))
                return;

        iterator = get_iterator_set(stmt->iterator_pre_statement);
        iter_set = expr_to_var(iterator);
        iterator = get_iterator_tested(stmt->iterator_pre_condition);
        iter_tested = expr_to_var(iterator);
        if (!iter_set || !iter_tested)
                goto free;
        if (strcmp(iter_set, iter_tested))
                goto free;

        /* smatch doesn't handle loops correctly so this silences some
         * false positives.
         */
        if (right_side_changes(stmt->iterator_pre_condition))
                goto free;

        if (implied_condition_false(stmt->iterator_pre_condition))
                sm_warning("we never enter this loop");

free:
        free_string(iter_set);
        free_string(iter_tested);
}

void check_bogus_loop(int id)
{
        my_id = id;
        add_hook(&match_loop, PRELOOP_HOOK);
}
```

```
**********************************************************
    2681 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_buffer_too_small_for_struct.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

STATE(too_small);

static void match_assign(struct expression *expr)
{
        struct symbol *left_type, *right_type;
        struct expression *size_expr;
        sval_t min_size;

        left_type = get_type(expr->left);
        if (!left_type || left_type->type != SYM_PTR)
                return;
        left_type = get_real_base_type(left_type);
        if (!left_type || left_type->type != SYM_STRUCT)
                return;

        right_type = get_type(expr->right);
        if (!right_type || right_type->type != SYM_PTR)
                return;
        right_type = get_real_base_type(right_type);
        if (!right_type)
                return;
        if (right_type != &void_ctype && type_bits(right_type) != 8)
                return;

        size_expr = get_size_variable(expr->right);
        if (!size_expr)
                return;

        get_absolute_min(size_expr, &min_size);
        if (min_size.value >= type_bytes(left_type))
                return;

        set_state_expr(my_id, expr->left, &too_small);
}

static void match_dereferences(struct expression *expr)
{
        struct symbol *left_type;
        struct expression *right;
```



```
        struct smatch_state *state;
        char *name;
        struct expression *size_expr;
        sval_t min_size;

        if (expr->type != EXPR_PREOP)
                return;

        expr = strip_expr(expr->unop);
        state = get_state_expr(my_id, expr);
        if (state != &too_small)
                return;

        left_type = get_type(expr);
        if (!left_type || left_type->type != SYM_PTR)
                return;
        left_type = get_real_base_type(left_type);
        if (!left_type || left_type->type != SYM_STRUCT)
                return;

        right = get_assigned_expr(expr);
        size_expr = get_size_variable(right);
        if (!size_expr)
                return;

        get_absolute_min(size_expr, &min_size);
        if (min_size.value >= type_bytes(left_type))
                return;

        name = expr_to_str(right);
        sm_warning("is '%s' large enough for 'struct %s'? %s", name, left_type->
        free_string(name);
        set_state_expr(my_id, expr, &undefined);
}

void check_buffer_too_small_for_struct(int id)
{
        my_id = id;

        add_hook(&match_assign, ASSIGNMENT_HOOK);
        add_hook(&match_dereferences, DEREF_HOOK);
}
```

```
**********************************************************
    1112 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_call_tree.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_call(struct expression *expr)
{
        char *fn_name;

        fn_name = expr_to_var(expr->fn);
        if (!fn_name)
                return;
        sm_prefix();
        sm_printf("info: func_call (");
        print_held_locks();
        sm_printf(") %s\n", fn_name);
        free_string(fn_name);
}

void check_call_tree(int id)
{
        if (!option_call_tree)
                return;
        my_id = id;
        add_hook(&match_call, FUNCTION_CALL_HOOK);
}
```

```
**********************************************************
    3409 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_capable.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

STATE(capable);

static int capable_id;
static int ns_capable_id;

static void match_capable(const char *fn, struct expression *expr, void *_param)
{
        struct expression *arg;
        sval_t sval;
        char buf[32];

        arg = get_argument_from_call_expr(expr->args, 0);
        if (!get_implied_value(arg, &sval))
                return;
        snprintf(buf, sizeof(buf), "%s", sval_to_str(sval));
        set_state(capable_id, buf, NULL, &capable);
}

static void match_ns_capable(const char *fn, struct expression *expr, void *_par
{
        struct expression *arg;
        sval_t sval;
        char buf[32];

        if (get_function() && strcmp(get_function(), "capable") == 0)
                return;

        arg = get_argument_from_call_expr(expr->args, 1);
        if (!get_implied_value(arg, &sval))
                return;
        snprintf(buf, sizeof(buf), "%s", sval_to_str(sval));
        set_state(ns_capable_id, buf, NULL, &capable);
}

static void save_call_info(struct expression *call)
{
        struct sm_state *sm;

        FOR_EACH_MY_SM(capable_id, __get_cur_stree(), sm) {
                if (sm->state == &capable)
```



```
                        sql_insert_caller_info(call, CAPABLE, 0, sm->name, "");
        } END_FOR_EACH_SM(sm);

        FOR_EACH_MY_SM(ns_capable_id, __get_cur_stree(), sm) {
                if (sm->state == &capable)
                        sql_insert_caller_info(call, NS_CAPABLE, 0, sm->name, ""
        } END_FOR_EACH_SM(sm);
}

static void save_return_info(int return_id, char *return_ranges, struct expressi
{
        struct sm_state *sm;

        FOR_EACH_MY_SM(capable_id, __get_cur_stree(), sm) {
                if (sm->state == &capable)
                        sql_insert_return_states(return_id, return_ranges,
                                                 CAPABLE, 0, sm->name, "");
        } END_FOR_EACH_SM(sm);

        FOR_EACH_MY_SM(ns_capable_id, __get_cur_stree(), sm) {
                if (sm->state == &capable)
                        sql_insert_return_states(return_id, return_ranges,
                                                 CAPABLE, 0, sm->name, "");
        } END_FOR_EACH_SM(sm);
}

static void set_db_capable(const char *name, struct symbol *sym, char *key, char
{
        char buf[32];

        snprintf(buf, sizeof(buf), "%s", key);
        set_state(capable_id, buf, NULL, &capable);
}

static void set_db_ns_capable(const char *name, struct symbol *sym, char *key, c
{
        char buf[32];

        snprintf(buf, sizeof(buf), "%s", key);
        set_state(ns_capable_id, buf, NULL, &capable);
}

void check_capable(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        capable_id = id;
        add_function_hook("capable", &match_capable, INT_PTR(0));

        add_hook(&save_call_info, FUNCTION_CALL_HOOK);
        add_split_return_callback(save_return_info);
        select_caller_info_hook(set_db_capable, CAPABLE);
}

void check_ns_capable(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        ns_capable_id = id;
        add_function_hook("ns_capable", &match_ns_capable, INT_PTR(0));
        select_caller_info_hook(set_db_ns_capable, NS_CAPABLE);
}
```

```
**********************************************************
    1598 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_cast_assign.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

static struct symbol *get_cast_type(struct expression *expr)
{
        if (!expr || expr->type != EXPR_PREOP || expr->op != '*')
                return NULL;
        expr = strip_parens(expr->unop);
        if (expr->type != EXPR_CAST)
                return NULL;
        return get_pointer_type(expr);
}

static void match_overflow(struct expression *expr)
{
        struct expression *ptr;
        struct symbol *type;
        int cast_size;
        int data_size;

        type = get_cast_type(expr->left);
        if (!type)
                return;
        cast_size = type_bytes(type);

        ptr = strip_expr(expr->left->unop);
        data_size = get_array_size_bytes_min(ptr);
        if (data_size <= 0)
                return;
        if (data_size >= cast_size)
                return;
        sm_warning("potential memory corrupting cast %d vs %d bytes",
               cast_size, data_size);
}

void check_cast_assign(int id)
{
        my_id = id;
        add_hook(&match_overflow, ASSIGNMENT_HOOK);
}
```

```
**********************************************************
    4036 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_check_deref.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is like check_deref_check.c except that it complains about code like:
 * if (a)
 *       a->foo = 42;
 * a->bar = 7;
 *
 * Of course, Smatch has complained about these for forever but the problem is
 * the old scripts were too messy and complicated and generated too many false
 * positives.
 *
 * This check is supposed to be simpler because it only looks for one kind of
 * null dereference bug instead of every kind.  It also gets rid of the false
 * positives caused by the checks that happen inside macros.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(null);
STATE(ok);

static void is_ok(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &ok);
}

static void check_dereference(struct expression *expr)
{
        struct sm_state *sm;
        struct sm_state *tmp;

        if (__in_fake_assign)
                return;

        expr = strip_expr(expr);

        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return;
```



```
        if (is_ignored(my_id, sm->name, sm->sym))
                return;
        if (implied_not_equal(expr, 0))
                return;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (tmp->state == &merged)
                        continue;
                if (tmp->state == &ok)
                        continue;
                if (tmp->state == &null) {
                        sm_error("we previously assumed '%s' could be null (see
                                tmp->name, tmp->line);
                        add_ignore(my_id, sm->name, sm->sym);
                        return;
                }
        } END_FOR_EACH_PTR(tmp);
}

static void check_dereference_name_sym(char *name, struct symbol *sym)
{
        struct sm_state *sm;
        struct sm_state *tmp;

        sm = get_sm_state(my_id, name, sym);
        if (!sm)
                return;
        if (is_ignored(my_id, sm->name, sm->sym))
                return;
        if (implied_not_equal_name_sym(name, sym, 0))
                return;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (tmp->state == &merged)
                        continue;
                if (tmp->state == &ok)
                        continue;
                if (tmp->state == &null) {
                        sm_error("we previously assumed '%s' could be null (see
                                tmp->name, tmp->line);
                        add_ignore(my_id, sm->name, sm->sym);
                        return;
                }
        } END_FOR_EACH_PTR(tmp);
}

static void match_dereferences(struct expression *expr)
{
        if (expr->type != EXPR_PREOP)
                return;
        check_dereference(expr->unop);
}

static void match_pointer_as_array(struct expression *expr)
{
        if (!is_array(expr))
                return;
        check_dereference(get_array_base(expr));
}

static void set_param_dereferenced(struct expression *call, struct expression *a
{
        struct symbol *sym;
        char *name;

        name = get_variable_from_key(arg, key, &sym);
```

```
127         if (!name || !sym)
128                 goto free;
129
130         check_dereference_name_sym(name, sym);
131 free:
132         free_string(name);
133 }
134
135 static void match_condition(struct expression *expr)
136 {
137         struct smatch_state *true_state = NULL;
138
139         if (get_macro_name(expr->pos))
140                 return;
141
142         if (!is_pointer(expr))
143                 return;
144
145         if (expr->type == EXPR_ASSIGNMENT) {
146                 match_condition(expr->right);
147                 match_condition(expr->left);
148         }
149
150         if (implied_not_equal(expr, 0))
151                 return;
152
153         if (get_state_expr(my_id, expr))
154                 true_state = &ok;
155
156         set_true_false_states_expr(my_id, expr, true_state, &null);
157 }
158
159 void check_check_deref(int id)
160 {
161         my_id = id;
162
163         add_modification_hook(my_id, &is_ok);
164         add_hook(&match_dereferences, DEREF_HOOK);
165         add_hook(&match_pointer_as_array, OP_HOOK);
166         select_return_implies_hook(DEREFERENCE, &set_param_dereferenced);
167         add_hook(&match_condition, CONDITION_HOOK);
168 }
```

```
**********************************************************
    1905 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_container_of.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Some macros don't return NULL pointers.  Complain if people
 * check the results for NULL because obviously the programmers
 * don't know what the pants they're doing.
 */

#include "smatch.h"

static int my_id;

STATE(non_null);

static void is_ok(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &undefined);
}

static void match_non_null(const char *fn, struct expression *expr, void *unused
{
        set_state_expr(my_id, expr->left, &non_null);
}

static void match_condition(struct expression *expr)
{
        if (__in_pre_condition)
                return;

        if (get_macro_name(expr->pos))
                return;

        if (get_state_expr(my_id, expr) == &non_null) {
                char *name;

                name = expr_to_var(expr);
                sm_warning("can '%s' even be NULL?", name);
                set_state_expr(my_id, expr, &undefined);
                free_string(name);
        }
}

void check_container_of(int id)
{
        if (option_project != PROJ_KERNEL)
```



```
                return;

        my_id = id;
        add_macro_assign_hook("container_of", &match_non_null, NULL);
        add_macro_assign_hook("list_first_entry", &match_non_null, NULL);
        add_function_assign_hook("nla_data", &match_non_null, NULL);
        add_modification_hook(my_id, &is_ok);
        add_hook(&match_condition, CONDITION_HOOK);
}
```

```
**********************************************************
    2933 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_continue_vs_break.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * If you have code like:
 * do {
 *    if (xxx)
 *        continue;
 * while (0);
 *
 * Then the continue is equivalent of a break.  So what was really intended?
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static struct statement_list *iterator_stack;

static int is_do_while_zero(struct statement *stmt)
{
        if (!stmt->iterator_post_condition)
                return 0;
        if (!is_zero(stmt->iterator_post_condition))
                return 0;
        return 1;
}

static void push_statement(struct statement_list **stack, struct statement *stmt
{
        add_ptr_list(stack, stmt);
}

static void pop_statement(struct statement_list **stack)
{
        delete_ptr_list_last((struct ptr_list **)stack);
}

static int inside_do_while_zero(void)
{
        struct statement *stmt;

        stmt = last_ptr_list((struct ptr_list *)iterator_stack);
        return !!stmt;
}
```



```
static int loop_is_macro(void)
{
        struct statement *stmt;

        stmt = last_ptr_list((struct ptr_list *)iterator_stack);
        if (!stmt)
                return 0;
        if (get_macro_name(stmt->iterator_post_condition->pos))
                return 1;
        return 0;
}

static void match_stmt(struct statement *stmt)
{
        if (stmt->type != STMT_ITERATOR)
                return;

        if (is_do_while_zero(stmt)) {
                push_statement(&iterator_stack, stmt);
        } else
                push_statement(&iterator_stack, NULL);
}

static void match_stmt_after(struct statement *stmt)
{
        if (stmt->type != STMT_ITERATOR)
                return;

        pop_statement(&iterator_stack);
}

static void match_inline_start(struct expression *expr)
{
        push_statement(&iterator_stack, NULL);
}

static void match_inline_end(struct expression *expr)
{
        pop_statement(&iterator_stack);
}

static void match_continue(struct statement *stmt)
{
        if (stmt->type != STMT_GOTO)
                return;

        if (!stmt->goto_label || stmt->goto_label->type != SYM_NODE)
                return;
        if (strcmp(stmt->goto_label->ident->name, "continue") != 0)
                return;
        if (!inside_do_while_zero())
                return;
        if (loop_is_macro())
                return;
        sm_warning("continue to end of do { ... } while(0); loop");
}

void check_continue_vs_break(int id)
{
        my_id = id;
        add_hook(&match_stmt, STMT_HOOK);
        add_hook(&match_stmt_after, STMT_HOOK_AFTER);
        add_hook(&match_inline_start, INLINE_FN_START);
        add_hook(&match_inline_end, INLINE_FN_END);
```

```
127         add_hook(&match_continue, STMT_HOOK);
128 }
```

```
**********************************************************
   20797 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_debug.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

int local_debug;
static int my_id;
char *trace_variable;

static void match_all_values(const char *fn, struct expression *expr, void *info
{
        struct stree *stree;

        stree = get_all_states_stree(SMATCH_EXTRA);
        __print_stree(stree);
        free_stree(&stree);
}

static void match_cur_stree(const char *fn, struct expression *expr, void *info)
{
        __print_cur_stree();
}

static void match_state(const char *fn, struct expression *expr, void *info)
{
        struct expression *check_arg, *state_arg;
        struct sm_state *sm;
        int found = 0;

        check_arg = get_argument_from_call_expr(expr->args, 0);
        if (check_arg->type != EXPR_STRING) {
                sm_error("the check_name argument to %s is supposed to be a stri
                return;
        }
        state_arg = get_argument_from_call_expr(expr->args, 1);
        if (!state_arg || state_arg->type != EXPR_STRING) {
                sm_error("the state_name argument to %s is supposed to be a stri
                return;
        }

        FOR_EACH_SM(__get_cur_stree(), sm) {
                if (strcmp(check_name(sm->owner), check_arg->string->data) != 0)
                        continue;
                if (strcmp(sm->name, state_arg->string->data) != 0)
                        continue;
                sm_msg("'%s' = '%s'", sm->name, sm->state->name);
                found = 1;
        } END_FOR_EACH_SM(sm);

        if (!found)
                sm_msg("%s '%s' not found", check_arg->string->data, state_arg->
}

static void match_states(const char *fn, struct expression *expr, void *info)
{
        struct expression *check_arg;
        struct sm_state *sm;
        int found = 0;

        check_arg = get_argument_from_call_expr(expr->args, 0);
        if (check_arg->type != EXPR_STRING) {
                sm_error("the check_name argument to %s is supposed to be a stri
                return;
        }

        FOR_EACH_SM(__get_cur_stree(), sm) {
                if (strcmp(check_name(sm->owner), check_arg->string->data) != 0)
                        continue;
                sm_msg("%s", show_sm(sm));
                found = 1;
        } END_FOR_EACH_SM(sm);

        if (found)
                return;

        if (!id_from_name(check_arg->string->data))
                sm_msg("invalid check name '%s'", check_arg->string->data);
        else
                sm_msg("%s: no states", check_arg->string->data);
}

static void match_print_value(const char *fn, struct expression *expr, void *inf
{
        struct stree *stree;
        struct sm_state *tmp;
        struct expression *arg_expr;

        arg_expr = get_argument_from_call_expr(expr->args, 0);
        if (arg_expr->type != EXPR_STRING) {
                sm_error("the argument to %s is supposed to be a string literal"
                return;
        }

        stree = __get_cur_stree();
        FOR_EACH_MY_SM(SMATCH_EXTRA, stree, tmp) {
                if (!strcmp(tmp->name, arg_expr->string->data))
                        sm_msg("%s = %s", tmp->name, tmp->state->name);
        } END_FOR_EACH_SM(tmp);
}

static void match_print_known(const char *fn, struct expression *expr, void *inf
{
        struct expression *arg;
        struct range_list *rl = NULL;
        char *name;
        int known = 0;
        sval_t sval;

        arg = get_argument_from_call_expr(expr->args, 0);
        if (get_value(arg, &sval))
```

```
127                 known = 1;

129         get_implied_rl(arg, &rl);

131         name = expr_to_str(arg);
132         sm_msg("known: '%s' = '%s'.  implied = '%s'", name, known ? sval_to_str(
133         free_string(name);
134 }

136 static void match_print_implied(const char *fn, struct expression *expr, void *i
137 {
138         struct expression *arg;
139         struct range_list *rl = NULL;
140         char *name;

142         arg = get_argument_from_call_expr(expr->args, 0);
143         get_implied_rl(arg, &rl);

145         name = expr_to_str(arg);
146         sm_msg("implied: %s = '%s'", name, show_rl(rl));
147         free_string(name);
148 }

150 static void match_real_absolute(const char *fn, struct expression *expr, void *i
151 {
152         struct expression *arg;
153         struct range_list *rl = NULL;
154         char *name;

156         arg = get_argument_from_call_expr(expr->args, 0);
157         get_real_absolute_rl(arg, &rl);

159         name = expr_to_str(arg);
160         sm_msg("real absolute: %s = '%s'", name, show_rl(rl));
161         free_string(name);
162 }

164 static void match_print_implied_min(const char *fn, struct expression *expr, voi
165 {
166         struct expression *arg;
167         sval_t sval;
168         char *name;

170         arg = get_argument_from_call_expr(expr->args, 0);
171         name = expr_to_str(arg);

173         if (get_implied_min(arg, &sval))
174                 sm_msg("implied min: %s = %s", name, sval_to_str(sval));
175         else
176                 sm_msg("implied min: %s = <unknown>", name);

178         free_string(name);
179 }

181 static void match_print_implied_max(const char *fn, struct expression *expr, voi
182 {
183         struct expression *arg;
184         sval_t sval;
185         char *name;

187         arg = get_argument_from_call_expr(expr->args, 0);
188         name = expr_to_str(arg);

190         if (get_implied_max(arg, &sval))
191                 sm_msg("implied max: %s = %s", name, sval_to_str(sval));
192         else
```



```
193                 sm_msg("implied max: %s = <unknown>", name);

195         free_string(name);
196 }

198 static void match_user_rl(const char *fn, struct expression *expr, void *info)
199 {
200         struct expression *arg;
201         struct range_list *rl;
202         char *name;

204         arg = get_argument_from_call_expr(expr->args, 0);
205         name = expr_to_str(arg);

207         get_user_rl(arg, &rl);
208         sm_msg("user rl: '%s' = '%s'", name, show_rl(rl));

210         free_string(name);
211 }

213 static void match_capped(const char *fn, struct expression *expr, void *info)
214 {
215         struct expression *arg;
216         char *name;

218         arg = get_argument_from_call_expr(expr->args, 0);
219         name = expr_to_str(arg);
220         sm_msg("'%s' = '%s'", name, is_capped(arg) ? "capped" : "not capped");
221         free_string(name);
222 }

224 static void match_print_hard_max(const char *fn, struct expression *expr, void *
225 {
226         struct expression *arg;
227         sval_t sval;
228         char *name;

230         arg = get_argument_from_call_expr(expr->args, 0);
231         name = expr_to_str(arg);

233         if (get_hard_max(arg, &sval))
234                 sm_msg("hard max: %s = %s", name, sval_to_str(sval));
235         else
236                 sm_msg("hard max: %s = <unknown>", name);

238         free_string(name);
239 }

241 static void match_print_fuzzy_max(const char *fn, struct expression *expr, void
242 {
243         struct expression *arg;
244         sval_t sval;
245         char *name;

247         arg = get_argument_from_call_expr(expr->args, 0);
248         name = expr_to_str(arg);

250         if (get_fuzzy_max(arg, &sval))
251                 sm_msg("fuzzy max: %s = %s", name, sval_to_str(sval));
252         else
253                 sm_msg("fuzzy max: %s = <unknown>", name);

255         free_string(name);
256 }

258 static void match_print_absolute(const char *fn, struct expression *expr, void *
```

```
259 {
260         struct expression *arg;
261         struct range_list *rl;
262         char *name;

264         arg = get_argument_from_call_expr(expr->args, 0);
265         name = expr_to_str(arg);

267         get_absolute_rl(arg, &rl);
268         sm_msg("absolute: %s = %s", name, show_rl(rl));

270         free_string(name);
271 }

273 static void match_print_absolute_min(const char *fn, struct expression *expr, vo
274 {
275         struct expression *arg;
276         sval_t sval;
277         char *name;

279         arg = get_argument_from_call_expr(expr->args, 0);
280         name = expr_to_str(arg);

282         if (get_absolute_min(arg, &sval))
283                 sm_msg("absolute min: %s = %s", name, sval_to_str(sval));
284         else
285                 sm_msg("absolute min: %s = <unknown>", name);

287         free_string(name);
288 }

290 static void match_print_absolute_max(const char *fn, struct expression *expr, vo
291 {
292         struct expression *arg;
293         sval_t sval;
294         char *name;

296         arg = get_argument_from_call_expr(expr->args, 0);
297         get_absolute_max(arg, &sval);

299         name = expr_to_str(arg);
300         sm_msg("absolute max: %s = %s", name, sval_to_str(sval));
301         free_string(name);
302 }

304 static void match_sval_info(const char *fn, struct expression *expr, void *info)
305 {
306         struct expression *arg;
307         sval_t sval;
308         char *name;

310         arg = get_argument_from_call_expr(expr->args, 0);
311         name = expr_to_str(arg);

313         if (!get_implied_value(arg, &sval)) {
314                 sm_msg("no sval for '%s'", name);
315                 goto free;
316         }

318         sm_msg("implied: %s %c%d ->value = %llx", name, sval_unsigned(sval) ? 'u
319 free:
320         free_string(name);
321 }

323 static void match_member_name(const char *fn, struct expression *expr, void *inf
324 {
```



```
325         struct expression *arg;
326         char *name, *member_name;

328         arg = get_argument_from_call_expr(expr->args, 0);
329         name = expr_to_str(arg);
330         member_name = get_member_name(arg);
331         sm_msg("member name: '%s => %s'", name, member_name);
332         free_string(member_name);
333         free_string(name);
334 }

336 static void print_possible(struct sm_state *sm)
337 {
338         struct sm_state *tmp;

340         sm_msg("Possible values for %s", sm->name);
341         FOR_EACH_PTR(sm->possible, tmp) {
342                 printf("%s\n", tmp->state->name);
343         } END_FOR_EACH_PTR(tmp);
344         sm_msg("===");
345 }

347 static void match_possible(const char *fn, struct expression *expr, void *info)
348 {
349         struct stree *stree;
350         struct sm_state *tmp;
351         struct expression *arg_expr;

353         arg_expr = get_argument_from_call_expr(expr->args, 0);
354         if (arg_expr->type != EXPR_STRING) {
355                 sm_error("the argument to %s is supposed to be a string literal"
356                 return;
357         }

359         stree = __get_cur_stree();
360         FOR_EACH_MY_SM(SMATCH_EXTRA, stree, tmp) {
361                 if (!strcmp(tmp->name, arg_expr->string->data))
362                         print_possible(tmp);
363         } END_FOR_EACH_SM(tmp);
364 }

366 static void match_strlen(const char *fn, struct expression *expr, void *info)
367 {
368         struct expression *arg;
369         struct range_list *rl;
370         char *name;

372         arg = get_argument_from_call_expr(expr->args, 0);
373         get_implied_strlen(arg, &rl);

375         name = expr_to_str(arg);
376         sm_msg("strlen: '%s' %s characters", name, show_rl(rl));
377         free_string(name);
378 }

380 static void match_buf_size(const char *fn, struct expression *expr, void *info)
381 {
382         struct expression *arg, *comp;
383         struct range_list *rl;
384         int elements, bytes;
385         char *name;
386         char buf[256] = "";
387         int n;
388         sval_t sval;

390         arg = get_argument_from_call_expr(expr->args, 0);
```

```
392         elements = get_array_size(arg);
393         bytes = get_array_size_bytes_max(arg);
394         rl = get_array_size_bytes_rl(arg);
395         comp = get_size_variable(arg);
396
397         name = expr_to_str(arg);
398         n = snprintf(buf, sizeof(buf), "buf size: '%s' %d elements, %d bytes", n
399         free_string(name);
400
401         if (!rl_to_sval(rl, &sval))
402                 n += snprintf(buf + n, sizeof(buf) - n, " (rl = %s)", show_rl(rl
403
404         if (comp) {
405                 name = expr_to_str(comp);
406                 snprintf(buf + n, sizeof(buf) - n, "[size_var=%s]", name);
407                 free_string(name);
408         }
409         sm_msg("%s", buf);
410 }
411
412 static void match_note(const char *fn, struct expression *expr, void *info)
413 {
414         struct expression *arg_expr;
415
416         arg_expr = get_argument_from_call_expr(expr->args, 0);
417         if (arg_expr->type != EXPR_STRING) {
418                 sm_error("the argument to %s is supposed to be a string literal"
419                 return;
420         }
421         sm_msg("%s", arg_expr->string->data);
422 }
423
424 static void print_related(struct sm_state *sm)
425 {
426         struct relation *rel;
427
428         if (!estate_related(sm->state))
429                 return;
430
431         sm_prefix();
432         sm_printf("%s: ", sm->name);
433         FOR_EACH_PTR(estate_related(sm->state), rel) {
434                 sm_printf("%s ", rel->name);
435         } END_FOR_EACH_PTR(rel);
436         sm_printf("\n");
437 }
438
439 static void match_dump_related(const char *fn, struct expression *expr, void *in
440 {
441         struct stree *stree;
442         struct sm_state *tmp;
443
444         stree = __get_cur_stree();
445         FOR_EACH_MY_SM(SMATCH_EXTRA, stree, tmp) {
446                 print_related(tmp);
447         } END_FOR_EACH_SM(tmp);
448 }
449
450 static void match_compare(const char *fn, struct expression *expr, void *info)
451 {
452         struct expression *one, *two;
453         char *one_name, *two_name;
454         int comparison;
455         char buf[16];
```



```
457         one = get_argument_from_call_expr(expr->args, 0);
458         two = get_argument_from_call_expr(expr->args, 1);
459
460         comparison = get_comparison(one, two);
461         if (!comparison)
462                 snprintf(buf, sizeof(buf), "<none>");
463         else
464                 snprintf(buf, sizeof(buf), "%s", show_special(comparison));
465
466         one_name = expr_to_str(one);
467         two_name = expr_to_str(two);
468
469         sm_msg("%s %s %s", one_name, buf, two_name);
470
471         free_string(one_name);
472         free_string(two_name);
473 }
474
475 static void match_debug_on(const char *fn, struct expression *expr, void *info)
476 {
477         option_debug = 1;
478 }
479
480 static void match_debug_check(const char *fn, struct expression *expr, void *inf
481 {
482         struct expression *arg;
483
484         arg = get_argument_from_call_expr(expr->args, 0);
485         if (!arg || arg->type != EXPR_STRING)
486                 return;
487         option_debug_check = arg->string->data;
488         sm_msg("arg = '%s'", option_debug_check);
489 }
490
491 static void match_debug_off(const char *fn, struct expression *expr, void *info)
492 {
493         option_debug_check = (char *)"";
494         option_debug = 0;
495 }
496
497 static void match_local_debug_on(const char *fn, struct expression *expr, void *
498 {
499         local_debug = 1;
500 }
501
502 static void match_local_debug_off(const char *fn, struct expression *expr, void
503 {
504         local_debug = 0;
505 }
506
507 static void match_debug_implied_on(const char *fn, struct expression *expr, void
508 {
509         option_debug_implied = 1;
510 }
511
512 static void match_debug_implied_off(const char *fn, struct expression *expr, voi
513 {
514         option_debug_implied = 0;
515 }
516
517 static void match_about(const char *fn, struct expression *expr, void *info)
518 {
519         struct expression *arg;
520         struct sm_state *sm;
521         char *name;
```

```
523         sm_msg("---- about ----");
524         match_print_implied(fn, expr, NULL);
525         match_buf_size(fn, expr, NULL);
526         match_strlen(fn, expr, NULL);
527         match_real_absolute(fn, expr, NULL);

529         arg = get_argument_from_call_expr(expr->args, 0);
530         name = expr_to_str(arg);
531         if (!name) {
532                 sm_msg("info: not a straight forward variable.");
533                 return;
534         }

536         FOR_EACH_SM(__get_cur_stree(), sm) {
537                 if (strcmp(sm->name, name) != 0)
538                         continue;
539                 sm_msg("%s", show_sm(sm));
540         } END_FOR_EACH_SM(sm);
541 }

543 static void match_intersection(const char *fn, struct expression *expr, void *in
544 {
545         struct expression *one, *two;
546         struct range_list *one_rl, *two_rl;
547         struct range_list *res;

549         one = get_argument_from_call_expr(expr->args, 0);
550         two = get_argument_from_call_expr(expr->args, 1);

552         get_absolute_rl(one, &one_rl);
553         get_absolute_rl(two, &two_rl);

555         res = rl_intersection(one_rl, two_rl);
556         sm_msg("'%s' intersect '%s' is '%s'", show_rl(one_rl), show_rl(two_rl),
557 }

559 static void match_type(const char *fn, struct expression *expr, void *info)
560 {
561         struct expression *one;
562         struct symbol *type;
563         char *name;

565         one = get_argument_from_call_expr(expr->args, 0);
566         type = get_type(one);
567         name = expr_to_str(one);
568         sm_msg("type of '%s' is: '%s'", name, type_to_str(type));
569         free_string(name);
570 }

572 static int match_type_rl_return(struct expression *call, void *unused, struct ra
573 {
574         struct expression *one, *two;
575         struct symbol *type;

577         one = get_argument_from_call_expr(call->args, 0);
578         type = get_type(one);

580         two = get_argument_from_call_expr(call->args, 1);
581         if (!two || two->type != EXPR_STRING) {
582                 sm_msg("expected: __smatch_type_rl(type, \"string\")");
583                 return 0;
584         }
585         call_results_to_rl(call, type, two->string->data, rl);
586         return 1;
587 }
```



```
589 static void print_left_right(struct sm_state *sm)
590 {
591         if (!sm)
592                 return;
593         if (!sm->left && !sm->right)
594                 return;

596         sm_printf("[ ");
597         if (sm->left)
598                 sm_printf("(%d: %s->'%s')", get_stree_id(sm->left->pool),  sm->l
599         else
600                 sm_printf(" - ");


603         print_left_right(sm->left);

605         if (sm->right)
606                 sm_printf("(%d: %s->'%s')", get_stree_id(sm->right->pool),  sm->
607         else
608                 sm_printf(" - ");

610         print_left_right(sm->right);
611 }

613 static void match_print_merge_tree(const char *fn, struct expression *expr, void
614 {
615         struct sm_state *sm;
616         struct expression *arg;
617         char *name;

619         arg = get_argument_from_call_expr(expr->args, 0);
620         name = expr_to_str(arg);

622         sm = get_sm_state_expr(SMATCH_EXTRA, arg);
623         if (!sm) {
624                 sm_msg("no sm state for '%s'", name);
625                 goto free;
626         }

628         sm_prefix();
629         sm_printf("merge tree: %s -> %s", name, sm->state->name);
630         print_left_right(sm);
631         sm_printf("\n");

633 free:
634         free_string(name);
635 }

637 static void match_print_stree_id(const char *fn, struct expression *expr, void *
638 {
639         sm_msg("stree_id %d", __stree_id);
640 }

642 static void match_mtag(const char *fn, struct expression *expr, void *info)
643 {
644         struct expression *arg;
645         char *name;
646         mtag_t tag = 0;

648         arg = get_argument_from_call_expr(expr->args, 0);
649         name = expr_to_str(arg);
650         get_mtag(arg, &tag);
651         sm_msg("mtag: '%s' => tag: %lld", name, tag);
652         free_string(name);
653 }
```

```
655 static void match_mtag_data_offset(const char *fn, struct expression *expr, void
656 {
657         struct expression *arg;
658         char *name;
659         mtag_t tag = 0;
660         int offset = -1;
661
662         arg = get_argument_from_call_expr(expr->args, 0);
663         name = expr_to_str(arg);
664         expr_to_mtag_offset(arg, &tag, &offset);
665         sm_msg("mtag: '%s' => tag: %lld, offset: %d", name, tag, offset);
666         free_string(name);
667 }
668
669 static void match_state_count(const char *fn, struct expression *expr, void *inf
670 {
671         sm_msg("state_count = %d\n", sm_state_counter);
672 }
673
674 static void match_mem(const char *fn, struct expression *expr, void *info)
675 {
676         show_sname_alloc();
677         show_ptrlist_alloc();
678         sm_msg("%lu pools", get_pool_count());
679         sm_msg("%d strees", unfree_stree);
680         show_smatch_state_alloc();
681         show_sm_state_alloc();
682 }
683
684 static void match_exit(const char *fn, struct expression *expr, void *info)
685 {
686         exit(0);
687 }
688
689 static struct stree *old_stree;
690 static void trace_var(struct statement *stmt)
691 {
692         struct sm_state *sm, *old;
693         int printed = 0;
694
695         if (!trace_variable)
696                 return;
697         if (__inline_fn)
698                 return;
699
700         FOR_EACH_SM(__get_cur_stree(), sm) {
701                 if (strcmp(sm->name, trace_variable) != 0)
702                         continue;
703                 old = get_sm_state_stree(old_stree, sm->owner, sm->name, sm->sym
704                 if (old && old->state == sm->state)
705                         continue;
706                 sm_msg("[%d] %s '%s': '%s' => '%s'", stmt->type,
707                        check_name(sm->owner),
708                        sm->name, old ? old->state->name : "<none>", sm->state->n
709                 printed = 1;
710         } END_FOR_EACH_SM(sm);
711
712         if (printed) {
713                 free_stree(&old_stree);
714                 old_stree = clone_stree(__get_cur_stree());
715         }
716 }
717
718 static void free_old_stree(struct symbol *sym)
719 {
720         free_stree(&old_stree);
```



```
721 }
722
723 void check_debug(int id)
724 {
725         my_id = id;
726         add_function_hook("__smatch_about", &match_about, NULL);
727         add_function_hook("__smatch_all_values", &match_all_values, NULL);
728         add_function_hook("__smatch_state", &match_state, NULL);
729         add_function_hook("__smatch_states", &match_states, NULL);
730         add_function_hook("__smatch_value", &match_print_value, NULL);
731         add_function_hook("__smatch_known", &match_print_known, NULL);
732         add_function_hook("__smatch_implied", &match_print_implied, NULL);
733         add_function_hook("__smatch_implied_min", &match_print_implied_min, NULL
734         add_function_hook("__smatch_implied_max", &match_print_implied_max, NULL
735         add_function_hook("__smatch_user_rl", &match_user_rl, NULL);
736         add_function_hook("__smatch_capped", &match_capped, NULL);
737         add_function_hook("__smatch_hard_max", &match_print_hard_max, NULL);
738         add_function_hook("__smatch_fuzzy_max", &match_print_fuzzy_max, NULL);
739         add_function_hook("__smatch_absolute", &match_print_absolute, NULL);
740         add_function_hook("__smatch_absolute_min", &match_print_absolute_min, NU
741         add_function_hook("__smatch_absolute_max", &match_print_absolute_max, NU
742         add_function_hook("__smatch_real_absolute", &match_real_absolute, NULL);
743         add_function_hook("__smatch_sval_info", &match_sval_info, NULL);
744         add_function_hook("__smatch_member_name", &match_member_name, NULL);
745         add_function_hook("__smatch_possible", &match_possible, NULL);
746         add_function_hook("__smatch_cur_stree", &match_cur_stree, NULL);
747         add_function_hook("__smatch_strlen", &match_strlen, NULL);
748         add_function_hook("__smatch_buf_size", &match_buf_size, NULL);
749         add_function_hook("__smatch_note", &match_note, NULL);
750         add_function_hook("__smatch_dump_related", &match_dump_related, NULL);
751         add_function_hook("__smatch_compare", &match_compare, NULL);
752         add_function_hook("__smatch_debug_on", &match_debug_on, NULL);
753         add_function_hook("__smatch_debug_check", &match_debug_check, NULL);
754         add_function_hook("__smatch_debug_off", &match_debug_off, NULL);
755         add_function_hook("__smatch_local_debug_on", &match_local_debug_on, NULL
756         add_function_hook("__smatch_local_debug_off", &match_local_debug_off, NU
757         add_function_hook("__smatch_debug_implied_on", &match_debug_implied_on,
758         add_function_hook("__smatch_debug_implied_off", &match_debug_implied_off
759         add_function_hook("__smatch_intersection", &match_intersection, NULL);
760         add_function_hook("__smatch_type", &match_type, NULL);
761         add_implied_return_hook("__smatch_type_rl_helper", &match_type_rl_return
762         add_function_hook("__smatch_merge_tree", &match_print_merge_tree, NULL);
763         add_function_hook("__smatch_stree_id", &match_print_stree_id, NULL);
764         add_function_hook("__smatch_mtag", &match_mtag, NULL);
765         add_function_hook("__smatch_mtag_data", &match_mtag_data_offset, NULL);
766         add_function_hook("__smatch_state_count", &match_state_count, NULL);
767         add_function_hook("__smatch_mem", &match_mem, NULL);
768         add_function_hook("__smatch_exit", &match_exit, NULL);
769
770         add_hook(free_old_stree, AFTER_FUNC_HOOK);
771         add_hook(trace_var, STMT_HOOK_AFTER);
772 }
```

```
**********************************************************
    2815 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_debug.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef __SMATCH_CHECK_DEBUG
#define __SMATCH_CHECK_DEBUG

static inline void __smatch_about(long var){}

static inline void __smatch_cur_stree(void){}
static inline void __smatch_all_values(void){}
static inline void __smatch_state(const char *check_name, const char *state_name
static inline void __smatch_states(const char *check_name){}
static inline void __smatch_value(const char *unused){}
static inline void __smatch_known(long long val){}
static inline void __smatch_implied(long long val){}
static inline void __smatch_implied_min(long long val){}
static inline void __smatch_implied_max(long long val){}
static inline void __smatch_user_rl(long long val){}
static inline void __smatch_capped(long long val){}

static inline void __smatch_hard_max(long long val){}
static inline void __smatch_fuzzy_max(long long val){}

static inline void __smatch_absolute(long long val){}
static inline void __smatch_absolute_min(long long val){}
static inline void __smatch_absolute_max(long long val){}
static inline void __smatch_real_absolute(long long val){}

static inline void __smatch_sval_info(long long val){}

static inline void __smatch_member_name(long long val){}

static inline void __smatch_possible(const char *unused){}
static inline void __smatch_print_value(const char *unused){}

static inline void __smatch_strlen(const void *buf){}
static inline void __smatch_buf_size(const void *buf){}

static inline void __smatch_note(const char *note){}

static inline void __smatch_dump_related(void){}

static inline void __smatch_compare(long long one, long long two){}

static inline void __smatch_debug_on(void){}
static inline void __smatch_debug_check(const char *check_name){}
static inline void __smatch_debug_off(void){}

static inline void __smatch_local_debug_on(void){}
static inline void __smatch_local_debug_off(void){}

static inline void __smatch_debug_implied_on(void){}
static inline void __smatch_debug_implied_off(void){}

static inline void __smatch_intersection(long long one, long long two){}
static inline void __smatch_type(long long one){}

static long long __smatch_val;
static inline long long __smatch_type_rl_helper(long long type, const char *str,
{
        return __smatch_val;
}
#define __smatch_type_rl(type, fmt...) __smatch_type_rl_helper((type)0, fmt)
```



```
#define __smatch_rl(fmt...) __smatch_type_rl(long long, fmt)

static inline void __smatch_bit_info(long long expr){}

static inline void __smatch_oops(unsigned long null_val){}

static inline void __smatch_merge_tree(long long var){}

static inline void __smatch_stree_id(void){}

static inline void __smatch_mtag(void *p){}
static inline void __smatch_mtag_data(long long arg){}
static inline void __smatch_exit(void){}

static inline void __smatch_state_count(void){}
static inline void __smatch_mem(void){}
#endif
```

```
**********************************************************
    7085 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_deref.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * There was a previous null dereference test but it was too confusing and
 * difficult to debug.  This test is much simpler in its goals and scope.
 *
 * This test only complains about:
 * 1) dereferencing uninitialized variables
 * 2) dereferencing variables which were assigned as null.
 * 3) dereferencing variables which were assigned a function the returns
 *    null.
 *
 * If we dereference something then we complain if any of those three
 * are possible.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

#define __GFP_NOFAIL 0x800

STATE(null);
STATE(ok);
STATE(uninitialized);

static struct smatch_state *alloc_my_state(const char *name)
{
        struct smatch_state *state;

        state = __alloc_smatch_state(0);
        state->name = name;
        return state;
}

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        return &ok;
}

static void is_ok(struct sm_state *sm, struct expression *mod_expr)
{
```



```
        set_state(my_id, sm->name, sm->sym, &ok);
}

static void check_dereference(struct expression *expr)
{
        struct sm_state *sm;
        struct sm_state *tmp;

        expr = strip_expr(expr);
        if (is_static(expr))
                return;
        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return;
        if (is_ignored(my_id, sm->name, sm->sym))
                return;
        if (implied_not_equal(expr, 0))
                return;
        if (is_impossible_path())
                return;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (tmp->state == &merged)
                        continue;
                if (tmp->state == &ok)
                        continue;
                add_ignore(my_id, sm->name, sm->sym);
                if (tmp->state == &null) {
                        if (option_spammy)
                                sm_error("potential NULL dereference '%s'.", tmp
                        return;
                }
                if (tmp->state == &uninitialized) {
                        if (option_spammy)
                                sm_error("potentially dereferencing uninitialize
                        return;
                }
                sm_error("potential null dereference '%s'.  (%s returns null)",
                        tmp->name, tmp->state->name);
                return;
        } END_FOR_EACH_PTR(tmp);
}

static void check_dereference_name_sym(char *name, struct symbol *sym)
{
        struct sm_state *sm;
        struct sm_state *tmp;

        sm = get_sm_state(my_id, name, sym);
        if (!sm)
                return;
        if (is_ignored(my_id, sm->name, sm->sym))
                return;
        if (implied_not_equal_name_sym(name, sym, 0))
                return;
        if (is_impossible_path())
                return;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (tmp->state == &merged)
                        continue;
                if (tmp->state == &ok)
                        continue;
                add_ignore(my_id, sm->name, sm->sym);
                if (tmp->state == &null) {
                        if (option_spammy)
```

```
127                                 sm_error("potential NULL dereference '%s'.", tmp
128                         return;
129                 }
130                 if (tmp->state == &uninitialized) {
131                         if (option_spammy)
132                                 sm_error("potentially dereferencing uninitialize
133                         return;
134                 }
135                 sm_error("potential null dereference '%s'.  (%s returns null)",
136                         tmp->name, tmp->state->name);
137                 return;
138         } END_FOR_EACH_PTR(tmp);
139 }

141 static void match_dereferences(struct expression *expr)
142 {
143         if (expr->type != EXPR_PREOP)
144                 return;
145         check_dereference(expr->unop);
146 }

148 static void match_pointer_as_array(struct expression *expr)
149 {
150         if (!is_array(expr))
151                 return;
152         check_dereference(get_array_base(expr));
153 }

155 static void set_param_dereferenced(struct expression *call, struct expression *a
156 {
157         struct symbol *sym;
158         char *name;

160         name = get_variable_from_key(arg, key, &sym);
161         if (!name || !sym)
162                 goto free;

164         check_dereference_name_sym(name, sym);
165 free:
166         free_string(name);
167 }

169 static void match_declarations(struct symbol *sym)
170 {
171         const char *name;

173         if ((get_base_type(sym))->type == SYM_ARRAY)
174                 return;

176         if (!sym->ident)
177                 return;
178         name = sym->ident->name;
179         if (!sym->initializer) {
180                 set_state(my_id, name, sym, &uninitialized);
181                 scoped_state(my_id, name, sym);
182         }
183 }

185 static void match_assign(struct expression *expr)
186 {
187         struct statement *stmt;

189         if (!is_zero(expr->right))
190                 return;

192         if (__in_fake_assign)
```



```
193                 return;

195         FOR_EACH_PTR_REVERSE(big_statement_stack, stmt) {
196                 if (stmt->type == STMT_DECLARATION)
197                         return;
198                 break;
199         } END_FOR_EACH_PTR_REVERSE(stmt);

201         set_state_expr(my_id, expr->left, &null);
202 }

204 static void match_assigns_address(struct expression *expr)
205 {
206         struct expression *right;

208         right = strip_expr(expr->right);
209         if (right->type != EXPR_PREOP || right->op != '&')
210                 return;
211         set_state_expr(my_id, right, &ok);
212 }

214 static void match_condition(struct expression *expr)
215 {
216         if (expr->type == EXPR_ASSIGNMENT) {
217                 match_condition(expr->right);
218                 match_condition(expr->left);
219         }
220         if (!get_state_expr(my_id, expr))
221                 return;
222         set_true_false_states_expr(my_id, expr, &ok, NULL);
223 }

225 static int called_with_no_fail(struct expression *call, int param)
226 {
227         struct expression *arg;
228         sval_t sval;

230         if (param == -1)
231                 return 0;
232         call = strip_expr(call);
233         if (call->type != EXPR_CALL)
234                 return 0;
235         arg = get_argument_from_call_expr(call->args, param);
236         if (get_value(arg, &sval) && (sval.uvalue & __GFP_NOFAIL))
237                 return 1;
238         return 0;
239 }

241 static void match_assign_returns_null(const char *fn, struct expression *expr, v
242 {
243         struct smatch_state *state;
244         int gfp_param = PTR_INT(_gfp);

246         if (called_with_no_fail(expr->right, gfp_param))
247                 return;
248         state = alloc_my_state(fn);
249         set_state_expr(my_id, expr->left, state);
250 }

252 static void register_allocation_funcs(void)
253 {
254         struct token *token;
255         const char *func;
256         int arg;

258         token = get_tokens_file("kernel.allocation_funcs_gfp");
```

```
259         if (!token)
260                 return;
261         if (token_type(token) != TOKEN_STREAMBEGIN)
262                 return;
263         token = token->next;
264         while (token_type(token) != TOKEN_STREAMEND) {
265                 if (token_type(token) != TOKEN_IDENT)
266                         return;
267                 func = show_ident(token->ident);
268                 token = token->next;
269                 if (token_type(token) == TOKEN_IDENT)
270                         arg = -1;
271                 else if (token_type(token) == TOKEN_NUMBER)
272                         arg = atoi(token->number);
273                 else
274                         return;
275                 add_function_assign_hook(func, &match_assign_returns_null, INT_P
276                 token = token->next;
277         }
278         clear_token_alloc();
279 }

281 void check_deref(int id)
282 {
283         my_id = id;

285         add_unmatched_state_hook(my_id, &unmatched_state);
286         add_modification_hook(my_id, &is_ok);
287         add_hook(&match_dereferences, DEREF_HOOK);
288         add_hook(&match_pointer_as_array, OP_HOOK);
289         select_return_implies_hook(DEREFERENCE, &set_param_dereferenced);
290         add_hook(&match_condition, CONDITION_HOOK);
291         add_hook(&match_declarations, DECLARATION_HOOK);
292         add_hook(&match_assign, ASSIGNMENT_HOOK);
293         add_hook(&match_assigns_address, ASSIGNMENT_HOOK);
294         if (option_project == PROJ_KERNEL)
295                 register_allocation_funcs();
296 }
```

```
**********************************************************
    2240 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_deref_check.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;

STATE(derefed);

static void underef(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &undefined);
}

static void match_dereference(struct expression *expr)
{
        if (__in_fake_assign)
                return;

        if (expr->type != EXPR_PREOP)
                return;
        expr = strip_expr(expr->unop);
        if (!is_pointer(expr))
                return;
        if (implied_not_equal(expr, 0))
                return;

        if (is_impossible_path())
                return;

        set_state_expr(my_id, expr, &derefed);
}

static void set_param_dereferenced(struct expression *call, struct expression *a
{
        struct symbol *sym;
        char *name;

        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        if (implied_not_equal_name_sym(name, sym, 0))
                goto free;
        set_state(my_id, name, sym, &derefed);
```



```
free:
        free_string(name);
}

static void match_condition(struct expression *expr)
{
        struct sm_state *sm;

        if (__in_pre_condition)
                return;

        if (get_macro_name(expr->pos))
                return;

        if (!is_pointer(expr))
                return;

        sm = get_sm_state_expr(my_id, expr);
        if (!sm || sm->state != &derefed)
                return;

        sm_warning("variable dereferenced before check '%s' (see line %d)", sm->
        set_state_expr(my_id, expr, &undefined);
}

void check_deref_check(int id)
{
        my_id = id;
        add_hook(&match_dereference, DEREF_HOOK);
        add_hook(&match_condition, CONDITION_HOOK);
        select_return_implies_hook(DEREFERENCE, &set_param_dereferenced);
        add_modification_hook(my_id, &underef);
}
```

```
**********************************************************
    3120 Fri Dec 21 15:00:02 2018
new/usr/src/tools/smatch/src/check_dereferences_param.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is an --info recipe.  The goal is to print a message for every parameter
 * which we can not avoid dereferencing.  This is maybe a bit restrictive but it
 * avoids some false positives.
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

STATE(derefed);
STATE(ignore);
STATE(param);

static void set_ignore(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state == &derefed)
                return;
        set_state(my_id, sm->name, sm->sym, &ignore);
}

static void match_function_def(struct symbol *sym)
{
        struct symbol *arg;
        int i;

        i = -1;
        FOR_EACH_PTR(sym->ctype.base_type->arguments, arg) {
                i++;
                if (!arg->ident)
                        continue;
                set_state(my_id, arg->ident->name, arg, &param);
        } END_FOR_EACH_PTR(arg);
}

static void check_deref(struct expression *expr)
{
        struct expression *tmp;
        struct sm_state *sm;

        tmp = get_assigned_expr(expr);
```



```
        if (tmp)
                expr = tmp;
        expr = strip_expr(expr);

        if (get_param_num(expr) < 0)
                return;

        if (param_was_set(expr))
                return;

        sm = get_sm_state_expr(my_id, expr);
        if (sm && slist_has_state(sm->possible, &ignore))
                return;
        set_state_expr(my_id, expr, &derefed);
}

static void match_dereference(struct expression *expr)
{
        if (expr->type != EXPR_PREOP)
                return;
        if (getting_address())
                return;
        check_deref(expr->unop);
}

static void set_param_dereferenced(struct expression *call, struct expression *a
{
        /* XXX FIXME: param_implies has more information now */
        if (strcmp(key, "$") != 0)
                return;
        check_deref(arg);
}

static void process_states(void)
{
        struct sm_state *tmp;
        int arg;
        const char *name;

        FOR_EACH_MY_SM(my_id, __get_cur_stree(), tmp) {
                if (tmp->state != &derefed)
                        continue;
                arg = get_param_num_from_sym(tmp->sym);
                if (arg < 0)
                        continue;
                name = get_param_name(tmp);
                if (!name)
                        continue;
                sql_insert_return_implies(DEREFERENCE, arg, name, "1");
        } END_FOR_EACH_SM(tmp);
}

static void match_pointer_as_array(struct expression *expr)
{
        if (!is_array(expr))
                return;
        check_deref(get_array_base(expr));
}

void check_dereferences_param(int id)
{
        my_id = id;

        add_hook(&match_function_def, FUNC_DEF_HOOK);

        add_hook(&match_dereference, DEREF_HOOK);
```

```
127         add_hook(&match_pointer_as_array, OP_HOOK);
128         select_return_implies_hook(DEREFERENCE, &set_param_dereferenced);
129         add_modification_hook(my_id, &set_ignore);

131         all_return_states_hook(&process_states);
132 }
```

```
**********************************************************
    3282 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_dev_queue_xmit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * According to an email on lkml you are not allowed to reuse the skb
 * passed to dev_queue_xmit()
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(do_not_use);

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &undefined);
}

static int valid_use(void)
{
        struct expression *tmp;
        int i = 0;
        int dot_ops = 0;

        FOR_EACH_PTR_REVERSE(big_expression_stack, tmp) {
                if (!i++)
                        continue;
                if (tmp->type == EXPR_PREOP && tmp->op == '(')
                        continue;
                if (tmp->op == '.' && !dot_ops++)
                        continue;
//              if (tmp->type == EXPR_POSTOP)
//                      return 1;
                if (tmp->type == EXPR_CALL && sym_name_is("kfree_skb", tmp->fn))
                        return 1;
                return 0;
        } END_FOR_EACH_PTR_REVERSE(tmp);
        return 0;
}

/* match symbol is expensive.  only turn it on after we match the xmit function
static int match_symbol_active;
static void match_symbol(struct expression *expr)
```



```
{
        struct sm_state *sm;
        char *name;

        sm = get_sm_state_expr(my_id, expr);
        if (!sm || !slist_has_state(sm->possible, &do_not_use))
                return;
        if (valid_use())
                return;
        name = expr_to_var(expr);
        sm_error("'%s' was already used up by dev_queue_xmit()", name);
        free_string(name);
}

static void match_kfree_skb(const char *fn, struct expression *expr, void *param
{
        struct expression *arg;

        arg = get_argument_from_call_expr(expr->args, 0);
        if (!arg)
                return;
        set_state_expr(my_id, arg, &undefined);
}

static void match_xmit(const char *fn, struct expression *expr, void *param)
{
        struct expression *arg;

        arg = get_argument_from_call_expr(expr->args, PTR_INT(param));
        if (!arg)
                return;
        set_state_expr(my_id, arg, &do_not_use);
        if (!match_symbol_active++) {
                add_hook(&match_symbol, SYM_HOOK);
                add_function_hook("kfree_skb", &match_kfree_skb, NULL);
        }
}

static void register_funcs_from_file(void)
{
        struct token *token;
        const char *func;
        int arg;

        token = get_tokens_file("kernel.dev_queue_xmit");
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                func = show_ident(token->ident);
                token = token->next;
                if (token_type(token) != TOKEN_NUMBER)
                        return;
                arg = atoi(token->number);
                add_function_hook(func, &match_xmit, INT_PTR(arg));
                token = token->next;
        }
        clear_token_alloc();
}

void check_dev_queue_xmit(int id)
{
```

```
127         if (option_project != PROJ_KERNEL)
128                 return;
129         my_id = id;
130         add_modification_hook(my_id, ok_to_use);
131         register_funcs_from_file();
132 }
```

```
**********************************************************
    2120 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_dma_mapping_error.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(positive);
STATE(ok);

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state != &ok)
                set_state(my_id, sm->name, sm->sym, &ok);
}

static void match_assign(const char *fn, struct expression *expr, void *unused)
{
        set_state_expr(my_id, expr->left, &positive);
}

static void match_condition(struct expression *expr)
{
        if (!get_state_expr(my_id, expr))
                return;
        /* If the variable is zero that's ok */
        set_true_false_states_expr(my_id, expr, NULL, &ok);
}

static void match_return(struct expression *ret_value)
{
        struct smatch_state *state;
        struct sm_state *sm;
        sval_t min;

        sm = get_sm_state_expr(my_id, ret_value);
        if (!sm)
                return;
        if (!slist_has_state(sm->possible, &positive))
                return;
        state = get_state_expr(SMATCH_EXTRA, ret_value);
        if (!state)
                return;
        if (!get_absolute_min(ret_value, &min))
```



```
                return;
        if (min.value == 0)
                return;
        sm_warning("dma_mapping_error() doesn't return an error code");
}

void check_dma_mapping_error(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        my_id = id;
        add_function_assign_hook("dma_mapping_error", &match_assign, NULL);
        add_function_assign_hook("pci_dma_mapping_error", &match_assign, NULL);
        add_hook(&match_condition, CONDITION_HOOK);
        add_hook(&match_return, RETURN_HOOK);
        add_modification_hook(my_id, &ok_to_use);
}
```

```
**********************************************************
    2159 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_dma_on_stack.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_dma_func(const char *fn, struct expression *expr, void *param)
{
        struct expression *arg;
        struct symbol *sym;
        char *name;

        arg = get_argument_from_call_expr(expr->args, PTR_INT(param));
        arg = strip_expr(arg);
        if (!arg)
                return;
        if (arg->type == EXPR_PREOP && arg->op == '&') {
                if (arg->unop->type != EXPR_SYMBOL)
                        return;
                name = expr_to_str(arg);
                sm_error("doing dma on the stack (%s)", name);
                free_string(name);
                return;
        }
        if (arg->type != EXPR_SYMBOL)
                return;
        sym = get_type(arg);
        if (!sym || sym->type != SYM_ARRAY)
                return;
        if (get_param_num(arg) >= 0)
                return;
        name = expr_to_var(arg);
        sm_error("doing dma on the stack (%s)", name);
        free_string(name);
}

static void register_funcs_from_file(void)
{
        struct token *token;
        const char *func;
        int arg;

        token = get_tokens_file("kernel.dma_funcs");
        if (!token)
                return;
```



```
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                func = show_ident(token->ident);
                token = token->next;
                if (token_type(token) != TOKEN_NUMBER)
                        return;
                arg = atoi(token->number);
                add_function_hook(func, &match_dma_func, INT_PTR(arg));
                token = token->next;
        }
        clear_token_alloc();
}

void check_dma_on_stack(int id)
{
        if (option_project != PROJ_KERNEL)
                return;
        my_id = id;
        register_funcs_from_file();
}
```

```
**********************************************************
    6047 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_double_checking.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#define _GNU_SOURCE
#include <string.h>
#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(checked);
STATE(modified);

struct stree *to_check;

static struct statement *get_cur_stmt(void)
{
        return last_ptr_list((struct ptr_list *)big_statement_stack);
}

static void set_modified(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &modified);
}

static struct expression *strip_condition(struct expression *expr)
{
        expr = strip_expr(expr);

        if (expr->type == EXPR_PREOP && expr->op == '!')
                return strip_condition(expr->unop);

        if (expr->type == EXPR_COMPARE &&
            (expr->op == SPECIAL_EQUAL ||
             expr->op == SPECIAL_NOTEQUAL)) {
                if (is_zero(expr->left))
                        return strip_condition(expr->right);
                if (is_zero(expr->right))
                        return strip_condition(expr->left);
        }

        return expr;
}

static int conditions_match(struct expression *cond, struct expression *prev)
{
```



```
        prev = strip_condition(prev);

        if (prev == cond)
                return 1;

        if (prev->type == EXPR_LOGICAL) {
                if (conditions_match(cond, prev->left) ||
                    conditions_match(cond, prev->right))
                        return 1;
        }

        return 0;
}

/*
 * People like to do "if (foo) { ... } else if (!foo) { ... }".  Don't
 * complain when they do that even though it is nonsense.
 */
static int is_obvious_else(struct expression *cond)
{
        struct statement *parent;
        struct expression *prev;

        if (!get_cur_stmt())
                return 0;
        parent = get_cur_stmt()->parent;
        if (!parent)
                return 0;

        if (parent->type != STMT_IF)
                return 0;

        if (!parent->if_false)
                return 0;
        if (parent->if_false != get_cur_stmt())
                return 0;

        prev = strip_condition(parent->if_conditional);

        return conditions_match(cond, prev);
}

static int name_means_synchronize(const char *name)
{
        if (!name)
                return 0;

        if (strcasestr(name, "wait"))
                return 1;
        if (strcasestr(name, "down"))
                return 1;
        if (strcasestr(name, "lock") && !strcasestr(name, "unlock"))
                return 1;
        if (strcasestr(name, "delay"))
                return 1;
        if (strcasestr(name, "schedule"))
                return 1;
        if (strcmp(name, "smp_rmb") == 0)
                return 1;
        if (strcmp(name, "mb") == 0)
                return 1;
        if (strcmp(name, "barrier") == 0)
                return 1;
        return 0;
}
```

```
127 static int previous_statement_was_synchronize(void)
128 {
129         struct statement *stmt;
130         struct position pos;
131         struct position prev_pos;
132         char *ident;

134         if (__prev_stmt) {
135                 prev_pos = __prev_stmt->pos;
136                 prev_pos.line -= 3;
137         } else {
138                 prev_pos = __cur_stmt->pos;
139                 prev_pos.line -= 5;
140         }

142         FOR_EACH_PTR_REVERSE(big_statement_stack, stmt) {
143                 if (stmt->pos.line < prev_pos.line)
144                         return 0;
145                 pos = stmt->pos;
146                 ident = get_macro_name(pos);
147                 if (name_means_synchronize(ident))
148                         return 1;
149                 ident = pos_ident(pos);
150                 if (!ident)
151                         continue;
152                 if (strcmp(ident, "if") == 0) {
153                         pos.pos += 4;
154                         ident = pos_ident(pos);
155                         if (!ident)
156                                 continue;
157                 }
158                 if (name_means_synchronize(ident))
159                         return 1;
160         } END_FOR_EACH_PTR_REVERSE(stmt);
161         return 0;
162 }

164 static void match_condition(struct expression *expr)
165 {
166         struct smatch_state *state;
167         sval_t dummy;
168         char *name;

170         if (inside_loop())
171                 return;

173         if (get_value(expr, &dummy))
174                 return;

176         if (get_macro_name(expr->pos))
177                 return;

179         state = get_stored_condition(expr);
180         if (!state || !state->data)
181                 return;
182         if (get_macro_name(((struct expression *)state->data)->pos))
183                 return;

185         /*
186          * we allow double checking for NULL because people do this all the time
187          * and trying to stop them is a losers' battle.
188          */
189         if (is_pointer(expr) && implied_condition_true(expr))
190                 return;

192         if (definitely_inside_loop()) {
```



```
193                 struct symbol *sym;

195                 if (__inline_fn)
196                         return;

198                 name = expr_to_var_sym(expr, &sym);
199                 if (!name)
200                         return;
201                 set_state_expr(my_id, expr, &checked);
202                 set_state_stree(&to_check, my_id, name, sym, &checked);
203                 free_string(name);
204                 return;
205         }

207         if (is_obvious_else(state->data))
208                 return;

210         /*
211          * It's common to test something, then take a lock and test if it is
212          * still true.
213          */
214         if (previous_statement_was_synchronize())
215                 return;

217         name = expr_to_str(expr);
218         sm_warning("we tested '%s' before and it was '%s'", name, state->name);
219         free_string(name);
220 }

222 int get_check_line(struct sm_state *sm)
223 {
224         struct sm_state *tmp;

226         FOR_EACH_PTR(sm->possible, tmp) {
227                 if (tmp->state == &checked)
228                         return tmp->line;
229         } END_FOR_EACH_PTR(tmp);

231         return get_lineno();
232 }

234 static void after_loop(struct statement *stmt)
235 {
236         struct sm_state *check, *sm;

238         if (!stmt || stmt->type != STMT_ITERATOR)
239                 return;
240         if (definitely_inside_loop())
241                 return;
242         if (__inline_fn)
243                 return;

245         FOR_EACH_SM(to_check, check) {
246                 continue;
247                 sm = get_sm_state(my_id, check->name, check->sym);
248                 continue;
249                 if (!sm)
250                         continue;
251                 if (slist_has_state(sm->possible, &modified))
252                         continue;

254                 sm_printf("%s:%d %s() ", get_filename(), get_check_line(sm), get
255                 sm_printf("warn: we tested '%s' already\n", check->name);
256         } END_FOR_EACH_SM(check);

258         free_stree(&to_check);
```

```
259 }

261 static void match_func_end(struct symbol *sym)
262 {
263         if (__inline_fn)
264                 return;
265         if (to_check)
266                 sm_msg("debug: odd...  found an function without an end.");
267         free_stree(&to_check);
268 }

270 void check_double_checking(int id)
271 {
272         my_id = id;

274         if (!option_spammy)
275                 return;

277         add_hook(&match_condition, CONDITION_HOOK);
278         add_modification_hook(my_id, &set_modified);
279         add_hook(after_loop, STMT_HOOK_AFTER);
280         add_hook(&match_func_end, AFTER_FUNC_HOOK);
281 }
```

```
**********************************************************
    2016 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_err_ptr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static int err_ptr = 0;
static int returns_null = 0;

static void match_err_ptr(struct expression *expr)
{
        expr = strip_expr(expr);
        if (!expr)
                return;
        if (expr->type != EXPR_CALL)
                return;

        if (expr->fn->type != EXPR_SYMBOL || !expr->fn->symbol)
                return;
        if (!strcmp(expr->fn->symbol->ident->name, "ERR_PTR"))
                err_ptr = 1;
}

extern int check_assigned_expr_id;
static void match_return(struct expression *ret_value)
{
        struct state_list *slist;
        struct sm_state *tmp;
        sval_t sval;

        if (__inline_fn)
                return;
        match_err_ptr(ret_value);
        slist = get_possible_states_expr(check_assigned_expr_id, ret_value);
        FOR_EACH_PTR(slist, tmp) {
                if (tmp->state == &undefined || tmp->state == &merged)
                        continue;
                match_err_ptr((struct expression *)tmp->state->data);
        } END_FOR_EACH_PTR(tmp);

        if (get_implied_value(ret_value, &sval)) {
                if (sval.value == 0)
                        returns_null = 1;
        }
```



```
}

static void match_end_func(struct symbol *sym)
{
        if (__inline_fn)
                return;
        if (err_ptr)
                sm_info("returns_err_ptr");
        err_ptr = 0;
        returns_null = 0;
}

void check_err_ptr(int id)
{
        if (option_project != PROJ_KERNEL)
                return;
        if (!option_info)
                return;

        my_id = id;
        add_hook(&match_return, RETURN_HOOK);
        add_hook(&match_end_func, END_FUNC_HOOK);
}
```

```
**********************************************************
    6668 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_err_ptr_deref.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(err_ptr);
STATE(checked);

static sval_t err_ptr_min = {
        .type = &int_ctype,
        {.value = -4095},
};

static sval_t err_ptr_max = {
        .type = &int_ctype,
        {.value = -1},
};

struct range_list *err_ptr_rl;

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state != &checked)
                set_state(my_id, sm->name, sm->sym, &checked);
}

static void check_is_err_ptr(struct expression *expr)
{
        struct sm_state *sm;
        struct range_list *rl;

        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return;

        if (!slist_has_state(sm->possible, &err_ptr))
                return;

        get_absolute_rl(expr, &rl);
        if (!possibly_true_rl(rl, SPECIAL_EQUAL, err_ptr_rl))
                return;
```



```
        sm_error("'%s' dereferencing possible ERR_PTR()", sm->name);
        set_state(my_id, sm->name, sm->sym, &checked);
}

static void match_returns_err_ptr(const char *fn, struct expression *expr,
                                  void *info)
{
        set_state_expr(my_id, expr->left, &err_ptr);
}

static void set_param_dereferenced(struct expression *call, struct expression *a
{
        struct sm_state *sm;
        struct smatch_state *estate;
        struct symbol *sym;
        char *name;

        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        sm = get_sm_state(my_id, name, sym);
        if (!sm)
                goto free;

        if (!slist_has_state(sm->possible, &err_ptr))
                goto free;

        estate = get_state(SMATCH_EXTRA, name, sym);
        if (!estate || !possibly_true_rl(estate_rl(estate), SPECIAL_EQUAL, err_p
                goto free;

        sm_error("'%s' dereferencing possible ERR_PTR()", sm->name);
        set_state(my_id, sm->name, sm->sym, &checked);

free:
        free_string(name);
}

static void match_checked(const char *fn, struct expression *call_expr,
                          struct expression *assign_expr, void *unused)
{
        struct expression *arg;

        arg = get_argument_from_call_expr(call_expr->args, 0);
        arg = strip_expr(arg);
        while (arg->type == EXPR_ASSIGNMENT)
                arg = strip_expr(arg->left);
        set_state_expr(my_id, arg, &checked);
}

static void match_err(const char *fn, struct expression *call_expr,
                          struct expression *assign_expr, void *unused)
{
        struct expression *arg;

        arg = get_argument_from_call_expr(call_expr->args, 0);
        arg = strip_expr(arg);
        while (arg->type == EXPR_ASSIGNMENT)
                arg = strip_expr(arg->left);
        set_state_expr(my_id, arg, &err_ptr);
}

static void match_dereferences(struct expression *expr)
{
        if (expr->type != EXPR_PREOP)
```

```
127                 return;
128         check_is_err_ptr(expr->unop);
129 }

131 static void match_kfree(const char *fn, struct expression *expr, void *_arg_nr)
132 {
133         int arg_nr = PTR_INT(_arg_nr);
134         struct expression *arg;

136         arg = get_argument_from_call_expr(expr->args, arg_nr);
137         check_is_err_ptr(arg);
138 }

140 static void match_condition(struct expression *expr)
141 {
142         if (expr->type == EXPR_ASSIGNMENT) {
143                 match_condition(expr->right);
144                 match_condition(expr->left);
145         }
146         if (!get_state_expr(my_id, expr))
147                 return;
148         /* If we know the variable is zero that means it's not an ERR_PTR */
149         set_true_false_states_expr(my_id, expr, NULL, &checked);
150 }

152 static void register_err_ptr_funcs(void)
153 {
154         struct token *token;
155         const char *func;

157         token = get_tokens_file("kernel.returns_err_ptr");
158         if (!token)
159                 return;
160         if (token_type(token) != TOKEN_STREAMBEGIN)
161                 return;
162         token = token->next;
163         while (token_type(token) != TOKEN_STREAMEND) {
164                 if (token_type(token) != TOKEN_IDENT)
165                         return;
166                 func = show_ident(token->ident);
167                 add_function_assign_hook(func, &match_returns_err_ptr, NULL);
168                 token = token->next;
169         }
170         clear_token_alloc();
171 }

173 static void match_err_ptr_positive_const(const char *fn, struct expression *expr
174 {
175         struct expression *arg;
176         sval_t sval;

178         arg = get_argument_from_call_expr(expr->args, 0);

180         if (!get_value(arg, &sval))
181                 return;
182         if (sval_is_positive(sval) && sval_cmp_val(sval, 0) != 0)
183                 sm_error("passing non negative %s to ERR_PTR", sval_to_str(sval)
184 }

186 static void match_err_ptr(const char *fn, struct expression *expr, void *unused)
187 {
188         struct expression *arg;
189         struct sm_state *sm;
190         struct sm_state *tmp;
191         sval_t tmp_min;
192         sval_t tmp_max;
```



```
193         sval_t min = sval_type_max(&llong_ctype);
194         sval_t max = sval_type_min(&llong_ctype);

196         arg = get_argument_from_call_expr(expr->args, 0);
197         sm = get_sm_state_expr(SMATCH_EXTRA, arg);
198         if (!sm)
199                 return;
200         FOR_EACH_PTR(sm->possible, tmp) {
201                 tmp_min = estate_min(tmp->state);
202                 if (!sval_is_a_min(tmp_min) && sval_cmp(tmp_min, min) < 0)
203                         min = tmp_min;
204                 tmp_max = estate_max(tmp->state);
205                 if (!sval_is_a_max(tmp_max) && sval_cmp(tmp_max, max) > 0)
206                         max = tmp_max;
207         } END_FOR_EACH_PTR(tmp);
208         if (sval_is_negative(min) && sval_cmp_val(min, -4095) < 0)
209                 sm_error("%s too low for ERR_PTR", sval_to_str(min));
210         if (sval_is_positive(max) && sval_cmp_val(max, 0) != 0)
211                 sm_error("passing non negative %s to ERR_PTR", sval_to_str(max))
212 }

214 void check_err_ptr_deref(int id)
215 {
216         if (option_project != PROJ_KERNEL)
217                 return;

219         my_id = id;
220         return_implies_state("IS_ERR", 0, 0, &match_checked, NULL);
221         return_implies_state("IS_ERR", 1, 1, &match_err, NULL);
222         return_implies_state("IS_ERR_OR_NULL", 0, 0, &match_checked, NULL);
223         return_implies_state("IS_ERR_OR_NULL", 1, 1, &match_err, NULL);
224         return_implies_state("PTR_RET", 0, 0, &match_checked, NULL);
225         return_implies_state("PTR_RET", -4096, -1, &match_err, NULL);
226         register_err_ptr_funcs();
227         add_hook(&match_dereferences, DEREF_HOOK);
228         add_function_hook("ERR_PTR", &match_err_ptr_positive_const, NULL);
229         add_function_hook("ERR_PTR", &match_err_ptr, NULL);
230         add_hook(&match_condition, CONDITION_HOOK);
231         add_modification_hook(my_id, &ok_to_use);
232         add_function_hook("kfree", &match_kfree, INT_PTR(0));
233         add_function_hook("brelse", &match_kfree, INT_PTR(0));
234         add_function_hook("kmem_cache_free", &match_kfree, INT_PTR(1));
235         add_function_hook("vfree", &match_kfree, INT_PTR(0));

237         err_ptr_rl = clone_rl_permanent(alloc_rl(err_ptr_min, err_ptr_max));

239         select_return_implies_hook(DEREFERENCE, &set_param_dereferenced);
240 }
```

```
**********************************************************
    2229 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_expects_err_ptr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;
static struct symbol *func_sym;

STATE(argument);
STATE(ok);

static void set_ok(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state != &ok)
                set_state(my_id, sm->name, sm->sym, &ok);
}

static void match_function_def(struct symbol *sym)
{
        struct symbol *arg;

        func_sym = sym;
        FOR_EACH_PTR(func_sym->ctype.base_type->arguments, arg) {
                if (!arg->ident) {
                        continue;
                }
                set_state(my_id, arg->ident->name, arg, &argument);
        } END_FOR_EACH_PTR(arg);
}

static int get_arg_num(struct expression *expr)
{
        struct smatch_state *state;
        struct symbol *arg;
        struct symbol *this_arg;
        int i;

        expr = strip_expr(expr);
        if (expr->type != EXPR_SYMBOL)
                return -1;
        this_arg = expr->symbol;

        state = get_state_expr(my_id, expr);
        if (!state || state != &argument)
                return -1;

```



```
        i = 0;
        FOR_EACH_PTR(func_sym->ctype.base_type->arguments, arg) {
                if (arg == this_arg)
                        return i;
                i++;
        } END_FOR_EACH_PTR(arg);

        return -1;
}

static void match_is_err(const char *fn, struct expression *expr, void *unused)
{
        struct expression *arg;
        int arg_num;

        arg = get_argument_from_call_expr(expr->args, 0);
        arg_num = get_arg_num(arg);
        if (arg_num < 0)
                return;
        sm_msg("info: expects ERR_PTR %d", arg_num);
}

void check_expects_err_ptr(int id)
{
        if (option_project != PROJ_KERNEL)
                return;
        if (!option_info)
                return;

        my_id = id;
        add_hook(&match_function_def, FUNC_DEF_HOOK);
        add_modification_hook(my_id, &set_ok);
        add_function_hook("IS_ERR", &match_is_err, NULL);
}
```

```
**********************************************************
    5954 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_free.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * check_memory() is getting too big and messy.
 *
 */

#include <string.h>
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(freed);
STATE(ok);

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state != &ok)
                set_state(my_id, sm->name, sm->sym, &ok);
}

static void pre_merge_hook(struct sm_state *sm)
{
        if (is_impossible_path())
                set_state(my_id, sm->name, sm->sym, &ok);
}

static int is_freed(struct expression *expr)
{
        struct sm_state *sm;

        sm = get_sm_state_expr(my_id, expr);
        if (sm && slist_has_state(sm->possible, &freed))
                return 1;
        return 0;
}

static void match_symbol(struct expression *expr)
{
        struct expression *parent;
        char *name;

        if (is_impossible_path())
```



```
                return;
        if (__in_fake_parameter_assign)
                return;

        parent = expr_get_parent_expr(expr);
        while (parent && parent->type == EXPR_PREOP && parent->op == '(')
                parent = expr_get_parent_expr(parent);
        if (parent && parent->type == EXPR_PREOP && parent->op == '&')
                return;

        if (!is_freed(expr))
                return;
        name = expr_to_var(expr);
        sm_warning("'%s' was already freed.", name);
        free_string(name);
}

static void match_dereferences(struct expression *expr)
{
        char *name;

        if (__in_fake_parameter_assign)
                return;

        if (expr->type != EXPR_PREOP)
                return;

        if (is_impossible_path())
                return;

        expr = strip_expr(expr->unop);
        if (!is_freed(expr))
                return;
        name = expr_to_var(expr);
        sm_error("dereferencing freed memory '%s'", name);
        set_state_expr(my_id, expr, &ok);
        free_string(name);
}

static int ignored_params[16];

static void set_ignored_params(struct expression *call)
{
        struct expression *arg;
        const char *p;
        int i;

        memset(&ignored_params, 0, sizeof(ignored_params));

        i = -1;
        FOR_EACH_PTR(call->args, arg) {
                i++;
                if (arg->type != EXPR_STRING)
                        continue;
                goto found;
        } END_FOR_EACH_PTR(arg);

        return;

found:
        i++;
        p = arg->string->data;
        while ((p = strchr(p, '%'))) {
                if (i >= ARRAY_SIZE(ignored_params))
                        return;
                p++;
```

```
127                 if (*p == '%') {
128                         p++;
129                         continue;
130                 }
131                 if (*p == '.')
132                         p++;
133                 if (*p == '*')
134                         i++;
135                 if (*p == 'p')
136                         ignored_params[i] = 1;
137                 i++;
138         }
139 }

141 static int is_free_func(struct expression *fn)
142 {
143         char *name;
144         int ret = 0;

146         name = expr_to_str(fn);
147         if (!name)
148                 return 0;
149         if (strstr(name, "free"))
150                 ret = 1;
151         free_string(name);

153         return ret;
154 }

156 static void match_call(struct expression *expr)
157 {
158         struct expression *arg;
159         char *name;
160         int i;

162         if (is_impossible_path())
163                 return;

165         set_ignored_params(expr);

167         i = -1;
168         FOR_EACH_PTR(expr->args, arg) {
169                 i++;
170                 if (!is_pointer(arg))
171                         continue;
172                 if (!is_freed(arg))
173                         continue;
174                 if (ignored_params[i])
175                         continue;

177                 name = expr_to_var(arg);
178                 if (is_free_func(expr->fn))
179                         sm_error("double free of '%s'", name);
180                 else
181                         sm_warning("passing freed memory '%s'", name);
182                 set_state_expr(my_id, arg, &ok);
183                 free_string(name);
184         } END_FOR_EACH_PTR(arg);
185 }

187 static void match_return(struct expression *expr)
188 {
189         char *name;

191         if (is_impossible_path())
192                 return;
```



```
193         if (__in_fake_parameter_assign)
194                 return;

196         if (!expr)
197                 return;
198         if (!is_freed(expr))
199                 return;

201         name = expr_to_var(expr);
202         sm_warning("returning freed memory '%s'", name);
203         set_state_expr(my_id, expr, &ok);
204         free_string(name);
205 }

207 static void match_free(const char *fn, struct expression *expr, void *param)
208 {
209         struct expression *arg;

211         if (is_impossible_path())
212                 return;

214         arg = get_argument_from_call_expr(expr->args, PTR_INT(param));
215         if (!arg)
216                 return;
217         if (is_freed(arg)) {
218                 char *name = expr_to_var(arg);

220                 sm_error("double free of '%s'", name);
221                 free_string(name);
222         }
223         set_state_expr(my_id, arg, &freed);
224 }

226 static void set_param_freed(struct expression *call, struct expression *arg, cha
227 {
228         struct symbol *sym;
229         char *name;

231         name = get_variable_from_key(arg, key, &sym);
232         if (!name || !sym)
233                 goto free;

235         set_state(my_id, name, sym, &freed);
236 free:
237         free_string(name);
238 }

240 int parent_is_free_var_sym(const char *name, struct symbol *sym)
241 {
242         char buf[256];
243         char *start;
244         char *end;
245         struct smatch_state *state;

247         if (option_project == PROJ_KERNEL)
248                 return parent_is_free_var_sym_strict(name, sym);

250         strncpy(buf, name, sizeof(buf) - 1);
251         buf[sizeof(buf) - 1] = '\0';

253         start = &buf[0];
254         while ((*start == '&'))
255                 start++;

257         while ((end = strrchr(start, '-'))) {
258                 *end = '\0';
```

```
259                 state = __get_state(my_id, start, sym);
260                 if (state == &freed)
261                         return 1;
262         }
263         return 0;
264 }

266 int parent_is_free(struct expression *expr)
267 {
268         struct symbol *sym;
269         char *var;
270         int ret = 0;

272         expr = strip_expr(expr);
273         var = expr_to_var_sym(expr, &sym);
274         if (!var || !sym)
275                 goto free;
276         ret = parent_is_free_var_sym(var, sym);
277 free:
278         free_string(var);
279         return ret;
280 }

282 void check_free(int id)
283 {
284         my_id = id;

286         if (option_project == PROJ_KERNEL) {
287                 /* The kernel use check_free_strict.c */
288                 return;
289         }

291         add_function_hook("free", &match_free, INT_PTR(0));

293         if (option_spammy)
294                 add_hook(&match_symbol, SYM_HOOK);
295         add_hook(&match_dereferences, DEREF_HOOK);
296         add_hook(&match_call, FUNCTION_CALL_HOOK);
297         add_hook(&match_return, RETURN_HOOK);

299         add_modification_hook(my_id, &ok_to_use);
300         select_return_implies_hook(PARAM_FREED, &set_param_freed);
301         add_pre_merge_hook(my_id, &pre_merge_hook);
302 }
```

```
**********************************************************
    6352 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_free_strict.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * check_memory() is getting too big and messy.
 *
 */

#include <string.h>
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(freed);
STATE(ok);

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state != &ok)
                set_state(my_id, sm->name, sm->sym, &ok);
}

static void pre_merge_hook(struct sm_state *sm)
{
        if (is_impossible_path())
                set_state(my_id, sm->name, sm->sym, &ok);
}

static int is_freed(struct expression *expr)
{
        struct sm_state *sm;

        sm = get_sm_state_expr(my_id, expr);
        if (sm && slist_has_state(sm->possible, &freed))
                return 1;
        return 0;
}

static void match_symbol(struct expression *expr)
{
        struct expression *parent;
        char *name;

        if (is_impossible_path())
```



```
                return;
        if (__in_fake_parameter_assign)
                return;

        parent = expr_get_parent_expr(expr);
        while (parent && parent->type == EXPR_PREOP && parent->op == '(')
                parent = expr_get_parent_expr(parent);
        if (parent && parent->type == EXPR_PREOP && parent->op == '&')
                return;

        if (!is_freed(expr))
                return;
        name = expr_to_var(expr);
        sm_warning("'%s' was already freed.", name);
        free_string(name);
}

static void match_dereferences(struct expression *expr)
{
        char *name;

        if (expr->type != EXPR_PREOP)
                return;

        if (is_impossible_path())
                return;
        if (__in_fake_parameter_assign)
                return;

        expr = strip_expr(expr->unop);
        if (!is_freed(expr))
                return;
        name = expr_to_var(expr);
        sm_error("dereferencing freed memory '%s'", name);
        set_state_expr(my_id, expr, &ok);
        free_string(name);
}

static int ignored_params[16];

static void set_ignored_params(struct expression *call)
{
        struct expression *arg;
        const char *p;
        int i;

        memset(&ignored_params, 0, sizeof(ignored_params));

        i = -1;
        FOR_EACH_PTR(call->args, arg) {
                i++;
                if (arg->type != EXPR_STRING)
                        continue;
                goto found;
        } END_FOR_EACH_PTR(arg);

        return;

found:
        i++;
        p = arg->string->data;
        while ((p = strchr(p, '%'))) {
                if (i >= ARRAY_SIZE(ignored_params))
                        return;
                p++;
                if (*p == '%') {
```

```
127                         p++;
128                         continue;
129                 }
130                 if (*p == '.')
131                         p++;
132                 if (*p == '*')
133                         i++;
134                 if (*p == 'p')
135                         ignored_params[i] = 1;
136                 i++;
137         }
138 }

140 static int is_free_func(struct expression *fn)
141 {
142         char *name;
143         int ret = 0;

145         name = expr_to_str(fn);
146         if (!name)
147                 return 0;
148         if (strstr(name, "free"))
149                 ret = 1;
150         free_string(name);

152         return ret;
153 }

155 static void match_call(struct expression *expr)
156 {
157         struct expression *arg;
158         char *name;
159         int i;

161         if (is_impossible_path())
162                 return;

164         set_ignored_params(expr);

166         i = -1;
167         FOR_EACH_PTR(expr->args, arg) {
168                 i++;
169                 if (!is_pointer(arg))
170                         continue;
171                 if (!is_freed(arg))
172                         continue;
173                 if (ignored_params[i])
174                         continue;

176                 name = expr_to_var(arg);
177                 if (is_free_func(expr->fn))
178                         sm_error("double free of '%s'", name);
179                 else
180                         sm_warning("passing freed memory '%s'", name);
181                 set_state_expr(my_id, arg, &ok);
182                 free_string(name);
183         } END_FOR_EACH_PTR(arg);
184 }

186 static void match_return(struct expression *expr)
187 {
188         char *name;

190         if (is_impossible_path())
191                 return;
```



```
193         if (!expr)
194                 return;
195         if (!is_freed(expr))
196                 return;

198         name = expr_to_var(expr);
199         sm_warning("returning freed memory '%s'", name);
200         set_state_expr(my_id, expr, &ok);
201         free_string(name);
202 }

204 static void match_free(const char *fn, struct expression *expr, void *param)
205 {
206         struct expression *arg;

208         if (is_impossible_path())
209                 return;

211         arg = get_argument_from_call_expr(expr->args, PTR_INT(param));
212         if (!arg)
213                 return;
214         if (is_freed(arg)) {
215                 char *name = expr_to_var(arg);

217                 sm_error("double free of '%s'", name);
218                 free_string(name);
219         }
220         set_state_expr(my_id, arg, &freed);
221 }

223 static void set_param_freed(struct expression *expr, int param, char *key, char
224 {
225         struct expression *arg;
226         char *name;
227         struct symbol *sym;
228         struct sm_state *sm;

230         while (expr->type == EXPR_ASSIGNMENT)
231                 expr = strip_expr(expr->right);
232         if (expr->type != EXPR_CALL)
233                 return;

235         arg = get_argument_from_call_expr(expr->args, param);
236         if (!arg)
237                 return;
238         name = get_variable_from_key(arg, key, &sym);
239         if (!name || !sym)
240                 goto free;

242         if (!is_impossible_path()) {
243                 sm = get_sm_state(my_id, name, sym);
244                 if (sm && slist_has_state(sm->possible, &freed)) {
245                         sm_warning("'%s' double freed", name);
246                         set_state(my_id, name, sym, &ok);  /* fixme: doesn't sil
247                 }
248         }

250         set_state(my_id, name, sym, &freed);
251 free:
252         free_string(name);
253 }

255 int parent_is_free_var_sym_strict(const char *name, struct symbol *sym)
256 {
257         char buf[256];
258         char *start;
```

```
259         char *end;
260         struct smatch_state *state;

262         strncpy(buf, name, sizeof(buf) - 1);
263         buf[sizeof(buf) - 1] = '\0';

265         start = &buf[0];
266         while ((*start == '&'))
267                 start++;

269         while ((end = strrchr(start, '-'))) {
270                 *end = '\0';
271                 state = __get_state(my_id, start, sym);
272                 if (state == &freed)
273                         return 1;
274         }
275         return 0;
276 }

278 int parent_is_free_strict(struct expression *expr)
279 {
280         struct symbol *sym;
281         char *var;
282         int ret = 0;

284         expr = strip_expr(expr);
285         var = expr_to_var_sym(expr, &sym);
286         if (!var || !sym)
287                 goto free;
288         ret = parent_is_free_var_sym_strict(var, sym);
289 free:
290         free_string(var);
291         return ret;
292 }

294 void check_free_strict(int id)
295 {
296         my_id = id;

298         if (option_project != PROJ_KERNEL)
299                 return;

301         add_function_hook("kfree", &match_free, INT_PTR(0));
302         add_function_hook("kmem_cache_free", &match_free, INT_PTR(1));

304         if (option_spammy)
305                 add_hook(&match_symbol, SYM_HOOK);
306         add_hook(&match_dereferences, DEREF_HOOK);
307         add_hook(&match_call, FUNCTION_CALL_HOOK);
308         add_hook(&match_return, RETURN_HOOK);

310         add_modification_hook_late(my_id, &ok_to_use);
311         add_pre_merge_hook(my_id, &pre_merge_hook);

313         select_return_states_hook(PARAM_FREED, &set_param_freed);
314 }
```

```
**********************************************************
    2363 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_freeing_devm.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

STATE(devm);

static int my_id;

static void match_assign(const char *fn, struct expression *expr, void *unused)
{
        set_state_expr(my_id, expr->left, &devm);
}

static void match_free_func(const char *fn, struct expression *expr, void *_arg)
{
        struct expression *arg_expr;
        int arg = PTR_INT(_arg);
        char *name;

        arg_expr = get_argument_from_call_expr(expr->args, arg);
        if (!get_state_expr(my_id, arg_expr))
                return;
        name = expr_to_str(arg_expr);
        sm_warning("passing devm_ allocated variable to kfree. '%s'", name);
        free_string(name);
}

static void register_funcs_from_file(void)
{
        struct token *token;
        const char *func;
        int arg;

        token = get_tokens_file("kernel.frees_argument");
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                func = show_ident(token->ident);
                token = token->next;
                if (token_type(token) != TOKEN_NUMBER)
```



```
                        return;
                arg = atoi(token->number);
                add_function_hook(func, &match_free_func, INT_PTR(arg));
                token = token->next;
        }
        clear_token_alloc();
}

void check_freeing_devm(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        my_id = id;

        add_function_assign_hook("devm_kmalloc", &match_assign, NULL);
        add_function_assign_hook("devm_kzalloc", &match_assign, NULL);
        add_function_assign_hook("devm_kcalloc", &match_assign, NULL);
        add_function_assign_hook("devm_kmalloc_array", &match_assign, NULL);


        add_function_hook("kfree", &match_free_func, INT_PTR(0));
        add_function_hook("krealloc", &match_free_func, INT_PTR(0));
        register_funcs_from_file();
}
```

```
**********************************************************
    1351 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_freeing_null.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_free(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg_expr;
        char *name;
        sval_t sval;

        arg_expr = get_argument_from_call_expr(expr->args, 0);
        if (!get_implied_value(arg_expr, &sval))
                return;
        if (sval.value != 0)
                return;
        name = expr_to_var(arg_expr);
        sm_warning("calling %s() when '%s' is always NULL.", fn, name);
        free_string(name);
}

void check_freeing_null(int id)
{
        my_id = id;
        if (!option_spammy)
                return;
        if (option_project == PROJ_KERNEL)
                add_function_hook("kfree", &match_free, NULL);
        else
                add_function_hook("free", &match_free, NULL);
}
```

```
**********************************************************
    3431 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_frees_argument.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This script is for finding functions like hcd_buffer_free() which free
 * their arguments.  After running it, add those functions to check_memory.c
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(freed);

static struct symbol *this_func;
static struct tracker_list *freed_args = NULL;

static void match_function_def(struct symbol *sym)
{
        this_func = sym;
}

static int is_arg(char *name, struct symbol *sym)
{
        struct symbol *arg;
        const char *arg_name;

        FOR_EACH_PTR(this_func->ctype.base_type->arguments, arg) {
                arg_name = (arg->ident?arg->ident->name:"-");
                if (sym == arg && !strcmp(name, arg_name))
                        return 1;
        } END_FOR_EACH_PTR(arg);
        return 0;
}

static void match_kfree(const char *fn, struct expression *expr, void *info)
{
        struct expression *tmp;
        struct symbol *sym;
        char *name;

        tmp = get_argument_from_call_expr(expr->args, 0);
        tmp = strip_expr(tmp);
        name = expr_to_var_sym(tmp, &sym);
        if (is_arg(name, sym)) {
```



```
                set_state(my_id, name, sym, &freed);
        }
        free_string(name);
}

static int return_count = 0;
static void match_return(struct expression *ret_value)
{
        struct stree *stree;
        struct sm_state *tmp;
        struct tracker *tracker;

        if (__inline_fn)
                return;

        if (!return_count) {
                stree = __get_cur_stree();
                FOR_EACH_MY_SM(my_id, stree, tmp) {
                        if (tmp->state == &freed)
                                add_tracker(&freed_args, my_id, tmp->name,
                                            tmp->sym);
                } END_FOR_EACH_SM(tmp);
        } else {
                FOR_EACH_PTR(freed_args, tracker) {
                        tmp = get_sm_state(my_id, tracker->name, tracker->sym);
                        if (tmp && tmp->state != &freed)
                                del_tracker(&freed_args, my_id, tracker->name,
                                            tracker->sym);
                } END_FOR_EACH_PTR(tracker);
        }
}

static void print_arg(struct symbol *sym)
{
        struct symbol *arg;
        int i = 0;

        FOR_EACH_PTR(this_func->ctype.base_type->arguments, arg) {
                if (sym == arg) {
                        sm_info("free_arg %s %d", get_function(), i);
                        return;
                }
                i++;
        } END_FOR_EACH_PTR(arg);
}

static void match_end_func(struct symbol *sym)
{
        if (__inline_fn)
                return;
        if (is_reachable())
                match_return(NULL);
}

static void match_after_func(struct symbol *sym)
{
        struct tracker *tracker;

        if (__inline_fn)
                return;

        FOR_EACH_PTR(freed_args, tracker) {
                print_arg(tracker->sym);
        } END_FOR_EACH_PTR(tracker);

        free_trackers_and_list(&freed_args);
```

```
127         return_count = 0;
128 }

130 void check_frees_argument(int id)
131 {
132         if (!option_info)
133                 return;

135         my_id = id;
136         add_hook(&match_function_def, FUNC_DEF_HOOK);
137         if (option_project == PROJ_KERNEL)
138                 add_function_hook("kfree", &match_kfree, NULL);
139         else
140                 add_function_hook("free", &match_kfree, NULL);
141         add_hook(&match_return, RETURN_HOOK);
142         add_hook(&match_end_func, END_FUNC_HOOK);
143         add_hook(&match_after_func, AFTER_FUNC_HOOK);
144 }
```

```
**********************************************************
    2983 Fri Dec 21 15:00:03 2018
new/usr/src/tools/smatch/src/check_frees_param.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This file is sort of like check_dereferences_param.c.  In theory the one
 * difference should be that the param is NULL it should still be counted as a
 * free.  But for now I don't handle that case.
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

STATE(freed);
STATE(ignore);
STATE(param);

static void set_ignore(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &ignore);
}

static void match_function_def(struct symbol *sym)
{
        struct symbol *arg;
        int i;

        i = -1;
        FOR_EACH_PTR(sym->ctype.base_type->arguments, arg) {
                i++;
                if (!arg->ident)
                        continue;
                set_state(my_id, arg->ident->name, arg, &param);
        } END_FOR_EACH_PTR(arg);
}

static void freed_variable(struct expression *expr)
{
        struct sm_state *sm;

        expr = strip_expr(expr);
        if (get_param_num(expr) < 0)
                return;
```



```
        sm = get_sm_state_expr(my_id, expr);
        if (sm && slist_has_state(sm->possible, &ignore))
                return;
        set_state_expr(my_id, expr, &freed);
}

static void match_free(const char *fn, struct expression *expr, void *param)
{
        struct expression *arg;

        arg = get_argument_from_call_expr(expr->args, PTR_INT(param));
        if (!arg)
                return;
        freed_variable(arg);
}

static void set_param_freed(struct expression *call, struct expression *arg, cha
{
        /* XXX FIXME: return_implies has been updated with more information */
        if (strcmp(key, "$") != 0)
                return;
        freed_variable(arg);
}

static void process_states(void)
{
        struct sm_state *sm;
        int param;
        const char *param_name;

        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
                if (sm->state != &freed)
                        continue;
                param = get_param_num_from_sym(sm->sym);
                if (param < 0)
                        continue;
                param_name = get_param_name(sm);
                if (!param_name)
                        continue;
                sql_insert_return_implies(PARAM_FREED, param, param_name, "1");
        } END_FOR_EACH_SM(sm);

}

void check_frees_param(int id)
{
        my_id = id;

        if (option_project == PROJ_KERNEL) {
                /* The kernel uses check_frees_param_strict.c */
                return;
        }

        add_hook(&match_function_def, FUNC_DEF_HOOK);

        add_function_hook("free", &match_free, INT_PTR(0));

        select_return_implies_hook(PARAM_FREED, &set_param_freed);
        add_modification_hook(my_id, &set_ignore);

        all_return_states_hook(&process_states);
}
```

```
**********************************************************
    3978 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_frees_param_strict.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This file is sort of like check_dereferences_param.c.  In theory the one
 * difference should be that the param is NULL it should still be counted as a
 * free.  But for now I don't handle that case.
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

STATE(freed);
STATE(ignore);

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        if (sm->state != &freed)
                return &undefined;
        if (parent_is_null_var_sym(sm->name, sm->sym))
                return &freed;
        return &undefined;
}

static void set_ignore(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &ignore);
}

static int counter_was_inced(struct expression *expr)
{
        char *name;
        struct symbol *sym;
        char buf[256];
        int ret = 0;

        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                goto free;

        snprintf(buf, sizeof(buf), "%s->users.counter", name);
        ret = was_inced(buf, sym);
free:
```



```
        free_string(name);
        return ret;
}

static void match_free(const char *fn, struct expression *expr, void *param)
{
        struct expression *arg, *tmp;
        int cnt = 0;

        arg = get_argument_from_call_expr(expr->args, PTR_INT(param));
        if (!arg)
                return;
        while ((tmp = get_assigned_expr(arg))) {
                arg = strip_expr(tmp);
                if (cnt++ > 5)
                        break;
        }

        if (get_param_num(arg) < 0)
                return;
        if (param_was_set(arg))
                return;
        if (strcmp(fn, "kfree_skb") == 0 && counter_was_inced(arg))
                return;

        set_state_expr(my_id, arg, &freed);
}

static void set_param_freed(struct expression *expr, int param, char *key, char
{
        struct expression *arg;
        char *name;
        struct symbol *sym;

        while (expr->type == EXPR_ASSIGNMENT)
                expr = strip_expr(expr->right);
        if (expr->type != EXPR_CALL)
                return;

        arg = get_argument_from_call_expr(expr->args, param);
        if (!arg)
                return;
        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;
        if (get_param_num_from_sym(sym) < 0)
                goto free;

        if (param_was_set_var_sym(name, sym))
                goto free;

        set_state(my_id, name, sym, &freed);
free:
        free_string(name);
}

static void param_freed_info(int return_id, char *return_ranges, struct expressi
{
        struct sm_state *sm;
        int param;
        const char *param_name;

        if (on_atomic_dec_path())
                return;

        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
```

```
127                 if (sm->state != &freed)
128                         continue;

130                 param = get_param_num_from_sym(sm->sym);
131                 if (param < 0)
132                         continue;

134                 param_name = get_param_name(sm);
135                 if (!param_name)
136                         continue;

138                 sql_insert_return_states(return_id, return_ranges, PARAM_FREED,
139                                          param, param_name, "");
140         } END_FOR_EACH_SM(sm);
141 }

143 void check_frees_param_strict(int id)
144 {
145         my_id = id;

147         if (option_project != PROJ_KERNEL)
148                 return;

150         add_function_hook("kfree", &match_free, INT_PTR(0));
151         add_function_hook("kmem_cache_free", &match_free, INT_PTR(1));
152         add_function_hook("kfree_skb", &match_free, INT_PTR(0));
153         add_function_hook("kfree_skbmem", &match_free, INT_PTR(0));
154         add_function_hook("dma_pool_free", &match_free, INT_PTR(1));
155         add_function_hook("spi_unregister_controller", &match_free, INT_PTR(0));

157         select_return_states_hook(PARAM_FREED, &set_param_freed);
158         add_modification_hook(my_id, &set_ignore);
159         add_split_return_callback(&param_freed_info);

161         add_unmatched_state_hook(my_id, &unmatched_state);
162 }
```

```
**********************************************************
    4773 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_get_user_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Looks for integers that we get from the user which can be attacked
 * with an integer overflow.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_max_id;
static int my_min_id;

STATE(capped);
STATE(user_data);

static void match_condition(struct expression *expr)
{
        struct smatch_state *left_max_true = NULL;
        struct smatch_state *left_max_false = NULL;
        struct smatch_state *right_max_true = NULL;
        struct smatch_state *right_max_false = NULL;

        struct smatch_state *left_min_true = NULL;
        struct smatch_state *left_min_false = NULL;
        struct smatch_state *right_min_true = NULL;
        struct smatch_state *right_min_false = NULL;

        if (expr->type != EXPR_COMPARE)
                return;

        switch (expr->op) {
        case '<':
        case SPECIAL_LTE:
        case SPECIAL_UNSIGNED_LT:
        case SPECIAL_UNSIGNED_LTE:
                left_max_true = &capped;
                right_max_false = &capped;
                right_min_true = &capped;
                left_min_false = &capped;
                break;
        case '>':
        case SPECIAL_GTE:
        case SPECIAL_UNSIGNED_GT:
```



```
        case SPECIAL_UNSIGNED_GTE:
                left_max_false = &capped;
                right_max_true = &capped;
                left_min_true = &capped;
                right_min_false = &capped;
                break;
        case SPECIAL_EQUAL:
                left_max_true = &capped;
                right_max_true = &capped;
                left_min_true = &capped;
                right_min_true = &capped;
                break;
        case SPECIAL_NOTEQUAL:
                left_max_false = &capped;
                right_max_false = &capped;
                left_min_false = &capped;
                right_min_false = &capped;
                break;
        default:
                return;
        }

        if (get_state_expr(my_max_id, expr->left)) {
                set_true_false_states_expr(my_max_id, expr->left, left_max_true,
                set_true_false_states_expr(my_min_id, expr->left, left_min_true,
        }
        if (get_state_expr(my_max_id, expr->right)) {
                set_true_false_states_expr(my_max_id, expr->right, right_max_tru
                set_true_false_states_expr(my_min_id, expr->right, right_min_tru
        }
}

static void match_normal_assign(struct expression *expr)
{
        if (get_state_expr(my_max_id, expr->left)) {
                set_state_expr(my_max_id, expr->left, &capped);
                set_state_expr(my_min_id, expr->left, &capped);
        }
}

static void match_assign(struct expression *expr)
{
        char *name;

        name = get_macro_name(expr->pos);
        if (!name || strcmp(name, "get_user") != 0) {
                match_normal_assign(expr);
                return;
        }
        name = expr_to_var(expr->right);
        if (!name || strcmp(name, "__val_gu") != 0)
                goto free;
        set_state_expr(my_max_id, expr->left, &user_data);
        set_state_expr(my_min_id, expr->left, &user_data);
free:
        free_string(name);
}

static void check_expr(struct expression *expr)
{
        struct sm_state *sm;
        sval_t max;
        sval_t sval;
        char *name;
        int overflow = 0;
        int underflow = 0;
```

```
128         sm = get_sm_state_expr(my_max_id, expr);
129         if (sm && slist_has_state(sm->possible, &user_data)) {
130                 if (!get_absolute_max(expr, &max) || sval_cmp_val(max, 20000) >
131                         overflow = 1;
132         }
133
134         sm = get_sm_state_expr(my_min_id, expr);
135         if (sm && slist_has_state(sm->possible, &user_data)) {
136                 if (!get_absolute_min(expr, &sval) ||
137                     (sval_is_negative(sval) && sval_cmp_val(sval, -20000) < 0))
138                         underflow = 1;
139         }
140
141         if (!overflow && !underflow)
142                 return;
143
144         name = expr_to_var_sym(expr, NULL);
145         if (overflow && underflow)
146                 sm_warning("check for integer over/underflow '%s'", name);
147         else if (underflow)
148                 sm_warning("check for integer underflow '%s'", name);
149         else
150                 sm_warning("check for integer overflow '%s'", name);
151         free_string(name);
152
153         set_state_expr(my_max_id, expr, &capped);
154         set_state_expr(my_min_id, expr, &capped);
155 }
156
157 static void match_binop(struct expression *expr)
158 {
159         if (expr->op == '^')
160                 return;
161         if (expr->op == '&')
162                 return;
163         if (expr->op == '|')
164                 return;
165         if (expr->op == SPECIAL_RIGHTSHIFT)
166                 return;
167         if (expr->op == SPECIAL_LEFTSHIFT)
168                 return;
169
170         check_expr(expr->left);
171         check_expr(expr->right);
172 }
173
174 void check_get_user_overflow(int id)
175 {
176         if (option_project != PROJ_KERNEL)
177                 return;
178         my_max_id = id;
179         add_hook(&match_condition, CONDITION_HOOK);
180         add_hook(&match_assign, ASSIGNMENT_HOOK);
181         add_hook(&match_binop, BINOP_HOOK);
182 }
183
184 void check_get_user_overflow2(int id)
185 {
186         my_min_id = id;
187 }
```

**********
*1610 Fri Dec 21 15:00:04 2018*
*new/usr/src/tools/smatch/src/check_gfp_dma.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

/* this is stolen from the kernel but it's totally fair use dude...  */
#define __GFP_DMA       (0x01u)
#define __GFP_HIGHMEM   (0x02u)
#define __GFP_DMA32     (0x04u)
#define __GFP_MOVABLE   (0x08u)
#define GFP_ZONEMASK    (__GFP_DMA|__GFP_HIGHMEM|__GFP_DMA32|__GFP_MOVABLE)

static void match_alloc(const char *fn, struct expression *expr, void *_arg)
{
        int arg_nr = PTR_INT(_arg);
        struct expression *arg_expr;
        sval_t sval;

        arg_expr = get_argument_from_call_expr(expr->args, arg_nr);
        if (!get_value(arg_expr, &sval))
                return;
        if (sval.uvalue == 0) /* GFP_NOWAIT */
                return;
        if (!(sval.uvalue & ~GFP_ZONEMASK))
                sm_error("no modifiers for allocation.");
}

void check_gfp_dma(int id)
{
        my_id = id;
        if (option_project != PROJ_KERNEL)
                return;
        add_function_hook("kmalloc", &match_alloc, INT_PTR(1));
        add_function_hook("kzalloc", &match_alloc, INT_PTR(1));
}
```

```
**********************************************************
    3442 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_held_dev.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This check is supposed to find bugs in reference counting using dev_hold()
 * and dev_put().
 *
 * When a device is first held, if an error happens later in the function
 * it needs to be released on all the error paths.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

STATE(held);
STATE(released);

static void match_dev_hold(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg_expr;

        arg_expr = get_argument_from_call_expr(expr->args, 0);
        set_state_expr(my_id, arg_expr, &held);
}

static void match_dev_put(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg_expr;

        arg_expr = get_argument_from_call_expr(expr->args, 0);
        set_state_expr(my_id, arg_expr, &released);
}

static void match_returns_held(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *unused)
{
        if (assign_expr)
                set_state_expr(my_id, assign_expr->left, &held);
}

static void match_returns_null(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *unused)
```



```
{
        if (assign_expr)
                set_state_expr(my_id, assign_expr->left, &released);
}

static void check_for_held(void)
{
        struct stree *stree;
        struct sm_state *tmp;

        stree = __get_cur_stree();
        FOR_EACH_MY_SM(my_id, stree, tmp) {
                if (slist_has_state(tmp->possible, &held)) {
                        sm_warning("'%s' held on error path.",
                                tmp->name);
                }
        } END_FOR_EACH_SM(tmp);
}

static void print_returns_held(struct expression *expr)
{
        struct sm_state *sm;

        if (!option_info)
                return;
        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return;
        if (slist_has_state(sm->possible, &held))
                sm_info("returned dev is held.");
}

static void match_return(struct expression *ret_value)
{
        print_returns_held(ret_value);
        if (!is_error_return(ret_value))
                return;
        check_for_held();
}

static void register_returns_held_funcs(void)
{
        struct token *token;
        const char *func;

        token = get_tokens_file("kernel.returns_held_funcs");
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                func = show_ident(token->ident);
                return_implies_state(func, valid_ptr_min, valid_ptr_max,
                                     &match_returns_held, NULL);
                return_implies_state(func, 0, 0, &match_returns_null,
                                     NULL);
                token = token->next;
        }
        clear_token_alloc();
}

void check_held_dev(int id)
{
```

```
127         if (option_project != PROJ_KERNEL)
128                 return;

130         my_id = id;
131         add_function_hook("dev_hold", &match_dev_hold, NULL);
132         add_function_hook("dev_put", &match_dev_put, NULL);
133         register_returns_held_funcs();
134         add_hook(&match_return, RETURN_HOOK);
135 }
```

```
**********************************************************
    7694 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_implicit_dependencies.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

/* If set, we ignore struct type symbols as implicit dependencies */
static int ignore_structs;

static struct symbol *cur_syscall;
/* note: cannot track return type and remove from implicit dependencies,
 * because every syscall returns a long, and we don't have a good way to know
 * whether or not this is a resource. The only example I can think of is open
 * returning a filedescriptor, so in the implicit dep parsing, we will just
 * blacklist struct fd --> file
 */
static struct symbol *cur_return_type;
static char *syscall_name;

static struct tracker_list *read_list;  // what fields does syscall branch on?
static struct tracker_list *write_list; // what fields does syscall modify?
static struct tracker_list *arg_list;   // what struct arguments does the syscal
static struct tracker_list *parsed_syscalls; // syscalls we have already checked

static inline void prefix(void)
{
        printf("%s:%d %s() ", get_filename(), get_lineno(), get_function());
}

static void match_syscall_definition(struct symbol *sym)
{
        struct symbol *arg;
        struct tracker *tracker;
        char *macro;
        char *name;
        int is_syscall = 0;

        macro = get_macro_name(sym->pos);
        if (macro &&
            (strncmp("SYSCALL_DEFINE", macro, strlen("SYSCALL_DEFINE")) == 0 ||
             strncmp("COMPAT_SYSCALL_DEFINE", macro, strlen("COMPAT_SYSCALL_DEFI
                is_syscall = 1;

        name = get_function();

        if (name && strncmp(name, "sys_", 4) == 0)
                is_syscall = 1;

        if (name && strncmp(name, "compat_sys_", 11) == 0)
                is_syscall = 1;

        if (!is_syscall)
                return;

        FOR_EACH_PTR(parsed_syscalls, tracker) {
                if (tracker->sym == sym) // don't re-parse
                        return;
        } END_FOR_EACH_PTR(tracker);

        syscall_name = name;
        cur_syscall = sym;
```



```
        cur_return_type = cur_func_return_type();
        if (cur_return_type && cur_return_type->ident)
                sm_msg("return type: %s\n", cur_return_type->ident->name);


        FOR_EACH_PTR(sym->ctype.base_type->arguments, arg) {
                // set_state(my_id, arg->ident->name, arg, &user_data_set);
                sm_msg("=======check_impl: arguments for call %s=========\n", sy
                if (arg->type == SYM_STRUCT)
                        arg = get_real_base_type(arg);
                if (cur_return_type && cur_return_type->ident)
                        sm_msg("arg type: %s\n", cur_return_type->ident->name);
                // add_tracker(&arg_list, my_id, member, arg);
                sm_msg("================================\n");
        } END_FOR_EACH_PTR(arg);
}

static void print_read_list(void)
{
    struct tracker *tracker;
    int i = 0;

    FOR_EACH_PTR(read_list, tracker) {
            if (i == 0)
                    sm_printf("%s read_list: [", syscall_name);
            sm_printf("%s, ", tracker->name);
            i++;
    } END_FOR_EACH_PTR(tracker);

    if (i > 0)
            sm_printf("]\n");
}

static void print_write_list(void)
{
        struct tracker *tracker;
        int i = 0;

        FOR_EACH_PTR(write_list, tracker) {
                if (i == 0)
                        sm_printf("%s write_list: [", syscall_name);
                sm_printf("%s, ", tracker->name);
                i++;
        } END_FOR_EACH_PTR(tracker);

        if (i > 0)
                sm_printf("]\n");
}

static void print_arg_list(void)
{
        struct tracker *tracker;
        int i = 0;

        FOR_EACH_PTR(write_list, tracker) {
                if (i == 0)
                        sm_printf("%s arg_list: [", syscall_name);
                sm_printf("%s, ", tracker->name);
                i++;
        } END_FOR_EACH_PTR(tracker);

        if (i > 0)
                sm_printf("]\n");
}
```

```
127 static void match_after_syscall(struct symbol *sym)
128 {
129         if (!cur_syscall || sym != cur_syscall)
130                 return;
131         // printf("\n"); prefix();
132         // printf("exiting scope of syscall %s\n", get_function());
133         // printf("------------------------\n");
134         print_read_list();
135         print_write_list();
136         print_arg_list();
137         free_trackers_and_list(&read_list);
138         free_trackers_and_list(&write_list);
139         free_trackers_and_list(&arg_list);
140         add_tracker(&parsed_syscalls, my_id, syscall_name, sym);
141         cur_syscall = NULL;
142         cur_return_type = NULL;
143         syscall_name = NULL;
144 }

146 static void print_read_member_type(struct expression *expr)
147 {
148         char *member;
149         struct symbol *sym;
150         struct symbol *member_sym;

152         member = get_member_name(expr);
153         if (!member)
154                 return;

156         sym = get_type(expr->deref);
157         member_sym = get_type(expr);

159         if (member_sym->type == SYM_PTR)
160                 member_sym = get_real_base_type(member_sym);

162         /*
163         if (member_sym->type == SYM_STRUCT)
164                 printf("found struct type %s\n", member);
165         else
166                 printf("found non-struct type %s with enum value%d\n", member, m
167         */

169         if (ignore_structs && member_sym->type == SYM_STRUCT) {
170                 // printf("ignoring %s\n", member);
171                 return;
172         }

174         add_tracker(&read_list, my_id, member, sym);
175         // sm_msg("info: uses %s", member);
176         // prefix();
177         // printf("info: uses %s\n", member);
178         free_string(member);
179 }

181 static void print_write_member_type(struct expression *expr)
182 {
183         char *member;
184         struct symbol *sym;
185         struct symbol *member_sym;

187         member = get_member_name(expr);
188         if (!member)
189                 return;

191         sym = get_type(expr->deref);
192         member_sym = get_type(expr);
```



```
194         if (member_sym->type == SYM_PTR)
195                 member_sym = get_real_base_type(member_sym);

197         /*
198         if (member_sym->type == SYM_STRUCT)
199                 printf("found struct type %s\n", member);
200         else
201                 printf("found non-struct type %s with enum value%d\n", member, m
202         */

204         if (ignore_structs && member_sym->type == SYM_STRUCT) {
205                 // printf("ignoring %s\n", member);
206                 return;
207         }

209         add_tracker(&write_list, my_id, member, sym);
210         free_string(member);
211 }

213 static void match_condition(struct expression *expr)
214 {
215         struct expression *arg;

217         if (!cur_syscall)
218                 return;

220         // prefix(); printf("-- condition found\n");

222         if (expr->type == EXPR_COMPARE ||
223             expr->type == EXPR_BINOP ||
224             expr->type == EXPR_LOGICAL ||
225             expr->type == EXPR_ASSIGNMENT ||
226             expr->type == EXPR_COMMA) {
227                 match_condition(expr->left);
228                 match_condition(expr->right);
229                 return;
230         }

232         if (expr->type == EXPR_CALL) {
233                 FOR_EACH_PTR(expr->args, arg) {
234                         // if we find deref in conditional call,
235                         // mark it as a read dependency
236                         print_read_member_type(arg);
237                 } END_FOR_EACH_PTR(arg);
238                 return;
239         }

241         print_read_member_type(expr);
242 }


245 /* when we are parsing an inline function and can no longer nest,
246  * assume that all struct fields passed to nested inline functions
247  * are read dependencies
248  */
249 static void match_call_info(struct expression *expr)
250 {
251         struct expression *arg;
252         int i;

254         if (!__inline_fn || !cur_syscall)
255                 return;

257         // prefix(); printf("fn: %s\n", expr->fn->symbol->ident->name);
```

```
259         i = 0;
260         FOR_EACH_PTR(expr->args, arg) {
261                 /*
262                    if (arg->type == EXPR_DEREF)
263                    printf("arg %d is deref\n", i);
264                  */
265                 print_read_member_type(arg);
266                 i++;
267         } END_FOR_EACH_PTR(arg);
268 }

270 static void match_assign_value(struct expression *expr)
271 {
272         if (!cur_syscall)
273                 return;
274         print_write_member_type(expr->left);
275 }

277 static void unop_expr(struct expression *expr)
278 {
279         if (!cur_syscall)
280                 return;

282         if (expr->op == SPECIAL_ADD_ASSIGN || expr->op == SPECIAL_INCREMENT ||
283             expr->op == SPECIAL_SUB_ASSIGN || expr->op == SPECIAL_DECREMENT ||
284             expr->op == SPECIAL_MUL_ASSIGN || expr->op == SPECIAL_DIV_ASSIGN ||
285             expr->op == SPECIAL_MOD_ASSIGN || expr->op == SPECIAL_AND_ASSIGN ||
286             expr->op == SPECIAL_OR_ASSIGN || expr->op == SPECIAL_XOR_ASSIGN ||
287             expr->op == SPECIAL_SHL_ASSIGN || expr->op == SPECIAL_SHR_ASSIGN)
288                 print_write_member_type(strip_expr(expr->unop));
289 }

291 void check_implicit_dependencies(int id)
292 {
293         my_id = id;
294         ignore_structs = 0;

296         if (option_project != PROJ_KERNEL)
297                 return;
298         if (!option_info)
299                 return;

301         add_hook(&match_syscall_definition, AFTER_DEF_HOOK);
302         add_hook(&match_after_syscall, AFTER_FUNC_HOOK);
303         add_hook(&match_condition, CONDITION_HOOK);
304         add_hook(&match_call_info, FUNCTION_CALL_HOOK);

306         /* hooks to track written fields */
307         add_hook(&match_assign_value, ASSIGNMENT_HOOK_AFTER);
308         add_hook(&unop_expr, OP_HOOK);
309 }
```

```
**********************************************************
    4270 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_implicit_dependencies_tester.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "smatch.h"
#include "linearize.h"

static int my_id;
static struct symbol *cur_syscall;

static const char *expression_type_name(enum expression_type type)
{
        static const char *expression_type_name[] = {
                [EXPR_VALUE] = "EXPR_VALUE",
                [EXPR_STRING] = "EXPR_STRING",
                [EXPR_SYMBOL] = "EXPR_SYMBOL",
                [EXPR_TYPE] = "EXPR_TYPE",
                [EXPR_BINOP] = "EXPR_BINOP",
                [EXPR_ASSIGNMENT] = "EXPR_ASSIGNMENT",
                [EXPR_LOGICAL] = "EXPR_LOGICAL",
                [EXPR_DEREF] = "EXPR_DEREF",
                [EXPR_PREOP] = "EXPR_PREOP",
                [EXPR_POSTOP] = "EXPR_POSTOP",
                [EXPR_CAST] = "EXPR_CAST",
                [EXPR_FORCE_CAST] = "EXPR_FORCE_CAST",
                [EXPR_IMPLIED_CAST] = "EXPR_IMPLIED_CAST",
                [EXPR_SIZEOF] = "EXPR_SIZEOF",
                [EXPR_ALIGNOF] = "EXPR_ALIGNOF",
                [EXPR_PTRSIZEOF] = "EXPR_PTRSIZEOF",
                [EXPR_CONDITIONAL] = "EXPR_CONDITIONAL",
                [EXPR_SELECT] = "EXPR_SELECT",
                [EXPR_STATEMENT] = "EXPR_STATEMENT",
                [EXPR_CALL] = "EXPR_CALL",
                [EXPR_COMMA] = "EXPR_COMMA",
                [EXPR_COMPARE] = "EXPR_COMPARE",
                [EXPR_LABEL] = "EXPR_LABEL",
                [EXPR_INITIALIZER] = "EXPR_INITIALIZER",
                [EXPR_IDENTIFIER] = "EXPR_IDENTIFIER",
                [EXPR_INDEX] = "EXPR_INDEX",
                [EXPR_POS] = "EXPR_POS",
                [EXPR_FVALUE] = "EXPR_FVALUE",
                [EXPR_SLICE] = "EXPR_SLICE",
                [EXPR_OFFSETOF] = "EXPR_OFFSETOF",
        };
        return expression_type_name[type] ?: "UNKNOWN_EXPRESSION_TYPE";
}

static inline void prefix() {
        printf("%s:%d %s() ", get_filename(), get_lineno(), get_function());
}

static void match_syscall_definition(struct symbol *sym)
{
        // struct symbol *arg;
        char *macro;
        char *name;
        int is_syscall = 0;

        macro = get_macro_name(sym->pos);
        if (macro &&
            (strncmp("SYSCALL_DEFINE", macro, strlen("SYSCALL_DEFINE")) == 0 ||
             strncmp("COMPAT_SYSCALL_DEFINE", macro, strlen("COMPAT_SYSCALL_DEFI
                is_syscall = 1;
```



```
        name = get_function();

        /*
        if (!option_no_db && get_state(my_id, "this_function", NULL) != &called)
                if (name && strncmp(name, "sys_", 4) == 0)
                        is_syscall = 1;
        }
        */

        /* Ignore compat_sys b/c syzkaller doesn't fuzz these?
        if (name && strncmp(name, "compat_sys_", 11) == 0)
                is_syscall = 1;
        */

        if (!is_syscall)
                return;
        printf("------------------------\n");
        printf("\nsyscall found: %s at: ", name);
        prefix(); printf("\n");
        cur_syscall = sym;

        /*
        FOR_EACH_PTR(sym->ctype.base_type->arguments, arg) {
                set_state(my_id, arg->ident->name, arg, &user_data_set);
        } END_FOR_EACH_PTR(arg);
        */
}

static void match_after_syscall(struct symbol *sym) {
    if (cur_syscall && sym == cur_syscall) {
        printf("\n"); prefix();
        printf("exiting scope of syscall %s\n", get_function());
        printf("------------------------\n");
        cur_syscall = NULL;
    }
}

static void print_member_type(struct expression *expr)
{
        char *member;

        member = get_member_name(expr);
        if (!member)
                return;
        // sm_msg("info: uses %s", member);
        prefix();
        printf("info: uses %s\n", member);
        free_string(member);
}

static void match_condition(struct expression *expr) {
    if (!cur_syscall)
        return;

    /*
    prefix();
    printf("found conditional %s on line %d\n", expression_type_name(expr->type)
    printf("expr_str: %s\n", expr_to_str(expr));
    */

    /*
    switch (expr->type) {
        case EXPR_COMPARE:
            match_condition(expr->left);
            match_condition(expr->right);
            break;
```

```
127         case EXPR_SYMBOL:
128             printf("symbol: %s\n", expr->symbol_name->name);
129             break;
130         case EXPR_CALL:
131             break;
132     }
133     */

135     prefix(); printf("-- condition found\n");

137     if (expr->type == EXPR_COMPARE || expr->type == EXPR_BINOP
138             || expr->type == EXPR_LOGICAL
139             || expr->type == EXPR_ASSIGNMENT
140             || expr->type == EXPR_COMMA) {
141             match_condition(expr->left);
142             match_condition(expr->right);
143             return;
144     }
145     print_member_type(expr);

147 }

149 static void match_function_call(struct expression *expr) {
150     if (!cur_syscall)
151         return;
152     prefix();
153     printf("function call %s\n", expression_type_name(expr->type));
154 }

156 void check_implicit_dependencies_tester(int id)
157 {
158     my_id = id;

160     if (option_project != PROJ_KERNEL)
161         return;

163     add_hook(&match_syscall_definition, AFTER_DEF_HOOK);
164     add_hook(&match_after_syscall, AFTER_FUNC_HOOK);
165     add_hook(&match_condition, CONDITION_HOOK);
166     add_hook(&match_function_call, FUNCTION_CALL_HOOK);
167 }
```

```
**********************************************************
    2638 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_impossible_mask.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

#if 0
static unsigned long long find_possible_bits(struct expression *expr)
{
        sval_t sval;
        unsigned long long ret;
        int set;
        int i;

        expr = strip_expr(expr);

        if (get_implied_value(expr, &sval))
                return sval.uvalue;

        if (expr->type == EXPR_BINOP && (expr->op == '&' || expr->op == '|')) {
                unsigned long long left, right;

                left = find_possible_bits(expr->left);
                if (!left)
                        return 0;
                right = find_possible_bits(expr->right);
                if (!right)
                        return 0;

                if (expr->op == '&')
                        return left & right;
                return left | right;
        }

        get_absolute_max(expr, &sval);
        ret = sval.value;

        set = false;
        for (i = 63; i >= 0; i--) {
                if (ret & 1 << i)
                        set = true;
                if (set)
                        ret |= 1 << i;
        }
        return ret;
```



```
}
#endif

static unsigned long long get_possible_bits(struct expression *expr)
{
        sval_t sval;

        expr = strip_expr(expr);
        if (expr->type != EXPR_BINOP)
                return 0;
        if (expr->op != '&')
                return 0;
        if (!get_implied_value(expr->right, &sval))
                return 0;

        return sval.uvalue;
}

static void match_condition(struct expression *expr)
{
        struct symbol *type;
        sval_t sval;
        unsigned long long left_mask, right_mask;
        char *str;

        type = get_type(expr);
        if (!type)
                type = &int_ctype;

        if (expr->type != EXPR_COMPARE)
                return;
        if (expr->op != SPECIAL_EQUAL && expr->op != SPECIAL_NOTEQUAL)
                return;

        if (!get_value(expr->right, &sval))
                return;
        right_mask = sval.uvalue;

        left_mask = get_possible_bits(expr->left);
        if (!left_mask)
                return;

        if (type_bits(type) < 64) {
                left_mask &= (1ULL << type_bits(type)) - 1;
                right_mask &= (1ULL << type_bits(type)) - 1;
        }

        if ((left_mask & right_mask) == right_mask)
                return;

        str = expr_to_str(expr);
        sm_warning("masked condition '%s' is always %s.", str,
                   expr->op == SPECIAL_EQUAL ? "false" : "true");
        free_string(str);
}

void check_impossible_mask(int id)
{
        my_id = id;

        add_hook(&match_condition, CONDITION_HOOK);
}
```

```
**********************************************************
    5869 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_indenting.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static struct string_list *ignored_macros;

static int in_ignored_macro(struct statement *stmt)
{
        const char *macro;
        char *tmp;

        macro = get_macro_name(stmt->pos);
        if (!macro)
                return 0;

        FOR_EACH_PTR(ignored_macros, tmp) {
                if (!strcmp(tmp, macro))
                        return 1;
        } END_FOR_EACH_PTR(tmp);
        return 0;
}

static int missing_curly_braces(struct statement *stmt)
{
        int inside_pos;

        if (stmt->pos.pos == __prev_stmt->pos.pos)
                return 0;

        if (__prev_stmt->type == STMT_IF) {
                if (__prev_stmt->if_true->type == STMT_COMPOUND)
                        return 0;
                inside_pos = __prev_stmt->if_true->pos.pos;
        } else if (__prev_stmt->type == STMT_ITERATOR) {
                if (!__prev_stmt->iterator_pre_condition)
                        return 0;
                if (__prev_stmt->iterator_statement->type == STMT_COMPOUND)
                        return 0;
                inside_pos = __prev_stmt->iterator_statement->pos.pos;
        } else {
                return 0;
        }
```



```
        if (stmt->pos.pos != inside_pos)
                return 0;

        sm_warning("curly braces intended?");
        return 1;
}

static int prev_lines_say_endif(struct statement *stmt)
{
        struct token *token;
        struct position pos = stmt->pos;
        int i;

        pos.pos = 2;

        for (i = 0; i < 4; i++) {
                pos.line--;
                token = pos_get_token(pos);
                if (token && token_type(token) == TOKEN_IDENT &&
                    strcmp(show_ident(token->ident), "endif") == 0)
                        return 1;
        }

        return 0;
}

static int is_pre_or_post_statement(struct statement *stmt)
{
        if (!stmt->parent)
                return 0;
        if (stmt->parent->type != STMT_ITERATOR)
                return 0;
        if (stmt->parent->iterator_pre_statement == stmt ||
            stmt->parent->iterator_post_statement == stmt)
                return 1;
        return 0;
}

/*
 * If we go out of position, then warn, but don't warn when we go back
 * into the correct position.
 */
static int orig_pos;

/*
 * If the code has two statements on the same line then don't complain
 * on the following line.  This is a bit of hack because it relies on the
 * quirk that we don't process nested inline functions.
 */
static struct position ignore_prev;
static struct position ignore_prev_inline;

static void match_stmt(struct statement *stmt)
{
        if (stmt != __cur_stmt)
                return;
        if (!__prev_stmt)
                return;

        if (prev_lines_say_endif(stmt))
                return;

        if (is_pre_or_post_statement(stmt))
                return;
        /* ignore empty statements if (foo) frob();; */
        if (stmt->type == STMT_EXPRESSION && !stmt->expression)
```

```
127                 return;
128         if (__prev_stmt->type == STMT_EXPRESSION && !__prev_stmt->expression)
129                 return;

131         if (__prev_stmt->type == STMT_LABEL || __prev_stmt->type == STMT_CASE)
132                 return;
133         /*
134          * This is sort of ugly.  The first statement after a case/label is
135          * special.  Probably we should handle this in smatch_flow.c so that
136          * this is not a special case.  Anyway it's like this:
137          * "foo: one++; two++;"  The code is on the same line.
138          * Also there is still a false positive here, if the first case
139          * statement has two statements on the same line.  I'm not sure what the
140          * deal is with that.
141          */
142         if (stmt->type == STMT_CASE) {
143                 if (__next_stmt &&
144                     __next_stmt->pos.line == stmt->case_statement->pos.line)
145                         ignore_prev = __next_stmt->pos;
146                 return;
147         }
148         if (stmt->type == STMT_LABEL) {
149                 if (__next_stmt &&
150                     __next_stmt->pos.line == stmt->label_statement->pos.line)
151                         ignore_prev = __next_stmt->pos;
152                 return;
153         }

155         if (missing_curly_braces(stmt))
156                 return;

158         if (stmt->pos.line == __prev_stmt->pos.line) {
159                 if (__inline_fn)
160                         ignore_prev_inline = stmt->pos;
161                 else
162                         ignore_prev = stmt->pos;
163                 return;
164         }
165         if (stmt->pos.pos == __prev_stmt->pos.pos)
166                 return;

168         /* some people like to line up their break and case statements. */
169         if (stmt->type == STMT_GOTO && stmt->goto_label &&
170             stmt->goto_label->type == SYM_NODE &&
171             strcmp(stmt->goto_label->ident->name, "break") == 0) {
172                 if (__next_stmt && __next_stmt->type == STMT_CASE &&
173                     (stmt->pos.line == __next_stmt->pos.line ||
174                      stmt->pos.pos == __next_stmt->pos.pos))
175                         return;
176                 /*
177                  * If we have a compound and the last statement is a break then
178                  * it's probably intentional.  This is most likely inside a
179                  * case statement.
180                  */
181                 if (!__next_stmt)
182                         return;
183         }

185         if (cmp_pos(__prev_stmt->pos, ignore_prev) == 0 ||
186             cmp_pos(__prev_stmt->pos, ignore_prev_inline) == 0)
187                 return;

189         if (in_ignored_macro(stmt))
190                 return;

192         if (stmt->pos.pos == orig_pos) {
```



```
193                 orig_pos = 0;
194                 return;
195         }
196         sm_warning("inconsistent indenting");
197         orig_pos = __prev_stmt->pos.pos;
198 }

200 static void match_end_func(void)
201 {
202         if (__inline_fn)
203                 return;
204         orig_pos = 0;
205 }

207 static void register_ignored_macros(void)
208 {
209         struct token *token;
210         char *macro;
211         char name[256];

213         snprintf(name, 256, "%s.ignore_macro_indenting", option_project_str);

215         token = get_tokens_file(name);
216         if (!token)
217                 return;
218         if (token_type(token) != TOKEN_STREAMBEGIN)
219                 return;
220         token = token->next;
221         while (token_type(token) != TOKEN_STREAMEND) {
222                 if (token_type(token) != TOKEN_IDENT)
223                         return;
224                 macro = alloc_string(show_ident(token->ident));
225                 add_ptr_list(&ignored_macros, macro);
226                 token = token->next;
227         }
228         clear_token_alloc();
229 }

231 void check_indenting(int id)
232 {
233         my_id = id;
234         add_hook(&match_stmt, STMT_HOOK);
235         add_hook(&match_end_func, END_FUNC_HOOK);
236         register_ignored_macros();
237 }
```

```
**********************************************************
    8296 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_index_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int loop_id;

STATE(loop_end);

static int definitely_just_used_as_limiter(struct expression *array, struct expr
{
        sval_t sval;
        struct expression *tmp;

        if (!get_implied_value(offset, &sval))
                return 0;
        if (get_array_size(array) != sval.value)
                return 0;

        tmp = array;
        while ((tmp = expr_get_parent_expr(tmp))) {
                if (tmp->type == EXPR_PREOP && tmp->op == '&')
                        return 1;
        }

        return 0;
}

static int fake_get_hard_max(struct expression *expr, sval_t *sval)
{
        struct range_list *implied_rl;

        if (!get_hard_max(expr, sval))
                return 0;

        /*
         * The problem is that hard_max doesn't care about minimums
         * properly.  So if you give it thing like:
         *      err = (-10)-(-1)
         *      __smatch_hard_max(-err);
         *
         * Then it returns s32max instead of 10.
         */

        if (get_implied_rl(expr, &implied_rl) &&
            sval_cmp(rl_max(implied_rl), *sval) < 0)
                *sval = rl_max(implied_rl);
        return 1;
}

static int get_the_max(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;

        if (get_hard_max(expr, sval)) {
                struct range_list *implied_rl;

                /*
                 * The problem is that hard_max doesn't care about minimums
                 * properly.  So if you give it thing like:
                 *      err = (-10)-(-1)
                 *      __smatch_hard_max(-err);
                 *
                 * Then it returns s32max instead of 10.
                 */

                if (get_implied_rl(expr, &implied_rl) &&
                    sval_cmp(rl_max(implied_rl), *sval) < 0)
                        *sval = rl_max(implied_rl);
                return 1;
        }
        if (!option_spammy)
                return 0;

        /* Fixme:  use fuzzy max */

        if (!get_user_rl(expr, &rl))
                return 0;
        if (rl_max(rl).uvalue > sval_type_max(rl_type(rl)).uvalue - 4 &&
            is_capped(expr))
                return 0;

        *sval = rl_max(rl);
        return 1;
}

static int common_false_positives(struct expression *array, sval_t max)
{
        char *name;
        int ret;

        name = expr_to_str(array);

        /* Smatch can't figure out glibc's strcmp __strcmp_cg()
         * so it prints an error every time you compare to a string
         * literal array with 4 or less chars.
         */
        if (name &&
            (strcmp(name, "__s1") == 0 || strcmp(name, "__s2") == 0)) {
                ret = 1;
                goto free;
        }

        /* Ugh... People are saying that Smatch still barfs on glibc strcmp()
         * functions.
         */
        if (array) {
                char *macro;
```

```
128                 /* why is this again??? */
129                 if (array->type == EXPR_STRING &&
130                     max.value == array->string->length) {
131                         ret = 1;
132                         goto free;
133                 }

135                 macro = get_macro_name(array->pos);
136                 if (macro && max.uvalue < 4 &&
137                     (strcmp(macro, "strcmp")  == 0 ||
138                      strcmp(macro, "strncmp") == 0 ||
139                      strcmp(macro, "streq")   == 0 ||
140                      strcmp(macro, "strneq")  == 0 ||
141                      strcmp(macro, "strsep")  == 0)) {
142                         ret = 1;
143                         goto free;
144                 }
145         }

147         /*
148          * passing WORK_CPU_UNBOUND is idiomatic but Smatch doesn't understand
149          * how it's used so it causes a bunch of false positives.
150          */
151         if (option_project == PROJ_KERNEL && name &&
152             strcmp(name, "__per_cpu_offset") == 0) {
153                 ret = 1;
154                 goto free;
155         }
156         ret = 0;

158 free:
159         free_string(name);
160         return ret;
161 }

163 static int is_subtract(struct expression *expr)
164 {
165         struct expression *tmp;
166         int cnt = 0;

168         expr = strip_expr(expr);
169         while ((tmp = get_assigned_expr(expr))) {
170                 expr = strip_expr(tmp);
171                 if (++cnt > 5)
172                         break;
173         }

175         if (expr->type == EXPR_BINOP && expr->op == '-')
176                 return 1;
177         return 0;
178 }

180 static int constraint_met(struct expression *array_expr, struct expression *offs
181 {
182         char *data_str, *required, *unmet;
183         int ret = 0;

185         data_str = get_constraint_str(array_expr);
186         if (!data_str)
187                 return 0;

189         required = get_required_constraint(data_str);
190         if (!required)
191                 goto free_data_str;
```



```
193         unmet = unmet_constraint(array_expr, offset);
194         if (!unmet)
195                 ret = 1;
196         free_string(unmet);
197         free_string(required);

199 free_data_str:
200         free_string(data_str);
201         return ret;
202 }


205 static int should_warn(struct expression *expr)
206 {
207         struct expression *array_expr;
208         struct range_list *abs_rl;
209         sval_t hard_max = { .type = &int_ctype, };
210         sval_t fuzzy_max = { .type = &int_ctype, };
211         int array_size;
212         struct expression *offset;
213         sval_t max;

215         expr = strip_expr(expr);
216         if (!is_array(expr))
217                 return 0;

219         if (is_impossible_path())
220                 return 0;
221         array_expr = get_array_base(expr);
222         array_size = get_array_size(array_expr);
223         if (!array_size || array_size == 1)
224                 return 0;

226         offset = get_array_offset(expr);
227         get_absolute_rl(offset, &abs_rl);
228         fake_get_hard_max(offset, &hard_max);
229         get_fuzzy_max(offset, &fuzzy_max);

231         if (!get_the_max(offset, &max))
232                 return 0;
233         if (array_size > max.value)
234                 return 0;
235         if (constraint_met(array_expr, offset))
236                 return 0;

238         if (array_size > rl_max(abs_rl).uvalue)
239                 return 0;

241         if (definitely_just_used_as_limiter(array_expr, offset))
242                 return 0;

244         array_expr = strip_expr(array_expr);
245         if (common_false_positives(array_expr, max))
246                 return 0;

248         if (impossibly_high_comparison(offset))
249                 return 0;

251         return 1;

253 }

255 static int is_because_of_no_break(struct expression *offset)
256 {
257         if (get_state_expr(loop_id, offset) == &loop_end)
258                 return 1;
```

```
259         return 0;
260 }

262 static void array_check(struct expression *expr)
263 {
264         struct expression *array_expr;
265         struct range_list *abs_rl;
266         struct range_list *user_rl = NULL;
267         sval_t hard_max = { .type = &int_ctype, };
268         sval_t fuzzy_max = { .type = &int_ctype, };
269         int array_size;
270         struct expression *array_size_value, *comparison;
271         struct expression *offset;
272         sval_t max;
273         char *name;
274         int no_break = 0;

276         if (!should_warn(expr))
277                 return;

279         expr = strip_expr(expr);
280         array_expr = get_array_base(expr);
281         array_size = get_array_size(array_expr);
282         offset = get_array_offset(expr);

284         /*
285          * Perhaps if the offset is out of bounds that means a constraint
286          * applies or maybe it means we are on an impossible path.  So test
287          * again based on that assumption.
288          *
289          */
290         array_size_value = value_expr(array_size);
291         comparison = compare_expression(offset, SPECIAL_GTE, array_size_value);
292         if (assume(comparison)) {
293                 if (!should_warn(expr)) {
294                         end_assume();
295                         return;
296                 }
297                 no_break = is_because_of_no_break(offset);
298                 end_assume();
299         }

301         get_absolute_rl(offset, &abs_rl);
302         get_user_rl(offset, &user_rl);
303         fake_get_hard_max(offset, &hard_max);
304         get_fuzzy_max(offset, &fuzzy_max);

306         array_expr = strip_expr(array_expr);
307         name = expr_to_str(array_expr);

309         if (user_rl)
310                 max = rl_max(user_rl);
311         else
312                 max = rl_max(abs_rl);

314         if (!option_spammy && is_subtract(offset))
315                 return;

317         if (no_break) {
318                 sm_error("buffer overflow '%s' %d <= %s (assuming for loop doesn
319                         name, array_size, sval_to_str(max));
320         } else if (user_rl) {
321                 sm_error("buffer overflow '%s' %d <= %s user_rl='%s'%s",
322                         name, array_size, sval_to_str(max), show_rl(user_rl),
323                         is_subtract(offset) ? " subtract" : "");
324         } else {
```



```
325                 sm_error("buffer overflow '%s' %d <= %s%s",
326                         name, array_size, sval_to_str(max),
327                         is_subtract(offset) ? " subtract" : "");
328         }

330         free_string(name);
331 }

333 void check_index_overflow(int id)
334 {
335         add_hook(&array_check, OP_HOOK);
336 }

338 static void match_condition(struct expression *expr)
339 {
340         struct statement *stmt;

342         if (expr->type != EXPR_COMPARE)
343                 return;
344         if (expr->op != '<' && expr->op != SPECIAL_UNSIGNED_LT)
345                 return;

347         stmt = expr_get_parent_stmt(expr);
348         if (!stmt || stmt->type != STMT_ITERATOR)
349                 return;

351         set_true_false_states_expr(loop_id, expr->left, NULL, &loop_end);
352 }

354 static void set_undefined(struct sm_state *sm, struct expression *mod_expr)
355 {
356         if (sm->state == &loop_end)
357                 set_state(loop_id, sm->name, sm->sym, &undefined);
358 }

360 void check_index_overflow_loop_marker(int id)
361 {
362         loop_id = id;

364         add_hook(&match_condition, CONDITION_HOOK);
365         add_modification_hook(loop_id, &set_undefined);
366 }
```

```
**********************************************************
    2491 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_info_leak.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(alloced);
STATE(string);

static char *my_get_variable(struct expression *expr, struct symbol **sym)
{
        char *name;

        name = expr_to_var_sym(expr, sym);
        free_string(name);
        if (!name || !*sym)
                return NULL;

        return (*sym)->ident->name;
}

static void match_kmalloc(const char *fn, struct expression *expr, void *unused)
{
        char *name;
        struct symbol *sym;

        name = my_get_variable(expr->left, &sym);
        if (!name)
                return;
        set_state(my_id, name, sym, &alloced);
}

static void match_strcpy(const char *fn, struct expression *expr, void *unused)
{
        struct expression *dest;
        char *name;
        struct symbol *sym;

        dest = get_argument_from_call_expr(expr->args, 0);
        name = my_get_variable(dest, &sym);
        if (!name || !sym)
                return;
        if (!get_state(my_id, name, sym))
                return;
```



```
        set_state(my_id, name, sym, &string);
}

static void match_copy_to_user(const char *fn, struct expression *expr, void *un
{
        struct expression *src;
        char *name;
        struct symbol *sym;
        struct sm_state *sm;

        src = get_argument_from_call_expr(expr->args, 1);
        name = my_get_variable(src, &sym);
        if (!name || !sym)
                return;
        sm = get_sm_state(my_id, name, sym);
        if (!sm || !slist_has_state(sm->possible, &string))
                return;
        name = expr_to_var(src);
        sm_warning("possible info leak '%s'", name);
        free_string(name);
}

void check_info_leak(int id)
{
        if (option_project != PROJ_KERNEL)
                return;
        my_id = id;
        add_function_assign_hook("kmalloc", &match_kmalloc, NULL);
        add_function_hook("strcpy", &match_strcpy, NULL);
        add_function_hook("strlcpy", &match_strcpy, NULL);
        add_function_hook("strlcat", &match_strcpy, NULL);
        add_function_hook("strncpy", &match_strcpy, NULL);
        add_function_hook("copy_to_user", &match_copy_to_user, NULL);
}
```

```
**********************************************************
    1894 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_input_free_device.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Don't call input_free_device() after calling
 * input_unregister_device()
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

STATE(no_free);
STATE(ok);

static int my_id;

static void match_assign(struct expression *expr)
{
        if (get_state_expr(my_id, expr->left)) {
                set_state_expr(my_id, expr->left, &ok);
        }
}

static void match_input_unregister(const char *fn, struct expression *expr, void
{
        struct expression *arg;

        arg = get_argument_from_call_expr(expr->args, 0);
        set_state_expr(my_id, arg, &no_free);
}

static void match_input_free(const char *fn, struct expression *expr, void *data
{
        struct expression *arg;
        struct sm_state *sm;

        arg = get_argument_from_call_expr(expr->args, 0);
        sm = get_sm_state_expr(my_id, arg);
        if (!sm)
                return;
        if (!slist_has_state(sm->possible, &no_free))
                return;
        sm_error("don't call input_free_device() after input_unregister_device()
}
```



```
void check_input_free_device(int id)
{
        my_id = id;
        if (option_project != PROJ_KERNEL)
                return;
        add_hook(&match_assign, ASSIGNMENT_HOOK);
        add_function_hook("input_unregister_device", &match_input_unregister, NU
        add_function_hook("input_free_device", &match_input_free, NULL);
}
```

```
**********************************************************
   11807 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_kernel.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is kernel specific stuff for smatch_extra.
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_extra.h"

static int implied_err_cast_return(struct expression *call, void *unused, struct
{
        struct expression *arg;

        arg = get_argument_from_call_expr(call->args, 0);
        if (!get_implied_rl(arg, rl))
                *rl = alloc_rl(ll_to_sval(-4095), ll_to_sval(-1));
        return 1;
}

static void hack_ERR_PTR(struct symbol *sym)
{
        struct symbol *arg;
        struct smatch_state *estate;
        struct range_list *after;
        sval_t low_error;
        sval_t minus_one;
        sval_t zero;

        low_error.type = &long_ctype;
        low_error.value = -4095;

        minus_one.type = &long_ctype;
        minus_one.value = -1;

        zero.type = &long_ctype;
        zero.value = 0;

        if (!sym || !sym->ident)
                return;
        if (strcmp(sym->ident->name, "ERR_PTR") != 0)
                return;

        arg = first_ptr_list((struct ptr_list *)sym->ctype.base_type->arguments)
        if (!arg || !arg->ident)
```



```
                return;

        estate = get_state(SMATCH_EXTRA, arg->ident->name, arg);
        if (!estate) {
                after = alloc_rl(low_error, minus_one);
        } else {
                after = rl_intersection(estate_rl(estate), alloc_rl(low_error, z
                if (rl_equiv(estate_rl(estate), after))
                        return;
        }
        set_state(SMATCH_EXTRA, arg->ident->name, arg, alloc_estate_rl(after));
}

static void match_param_valid_ptr(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *_param)
{
        int param = PTR_INT(_param);
        struct expression *arg;
        struct smatch_state *pre_state;
        struct smatch_state *end_state;

        arg = get_argument_from_call_expr(call_expr->args, param);
        pre_state = get_state_expr(SMATCH_EXTRA, arg);
        end_state = estate_filter_range(pre_state, ll_to_sval(-4095), ll_to_sval
        set_extra_expr_nomod(arg, end_state);
}

static void match_param_err_or_null(const char *fn, struct expression *call_expr
                        struct expression *assign_expr, void *_param)
{
        int param = PTR_INT(_param);
        struct expression *arg;
        struct range_list *rl;
        struct smatch_state *pre_state;
        struct smatch_state *end_state;

        arg = get_argument_from_call_expr(call_expr->args, param);
        pre_state = get_state_expr(SMATCH_EXTRA, arg);
        rl = alloc_rl(ll_to_sval(-4095), ll_to_sval(0));
        rl = rl_intersection(estate_rl(pre_state), rl);
        rl = cast_rl(estate_type(pre_state), rl);
        end_state = alloc_estate_rl(rl);
        set_extra_expr_nomod(arg, end_state);
}

static void match_not_err(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *unused)
{
        struct expression *arg;
        struct smatch_state *pre_state;
        struct smatch_state *new_state;

        arg = get_argument_from_call_expr(call_expr->args, 0);
        pre_state = get_state_expr(SMATCH_EXTRA, arg);
        new_state = estate_filter_range(pre_state, sval_type_min(&long_ctype), l
        set_extra_expr_nomod(arg, new_state);
}

static void match_err(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *unused)
{
        struct expression *arg;
        struct smatch_state *pre_state;
        struct smatch_state *new_state;

        arg = get_argument_from_call_expr(call_expr->args, 0);
```

```
127         pre_state = get_state_expr(SMATCH_EXTRA, arg);
128         new_state = estate_filter_range(pre_state, sval_type_min(&long_ctype), l
129         new_state = estate_filter_range(new_state, ll_to_sval(0), sval_type_max(
130         set_extra_expr_nomod(arg, new_state);
131 }

133 static void match_container_of_macro(const char *fn, struct expression *expr, vo
134 {
135         set_extra_expr_mod(expr->left, alloc_estate_range(valid_ptr_min_sval, va
136 }

138 static void match_container_of(struct expression *expr)
139 {
140         struct expression *right = expr->right;
141         char *macro;

143         /*
144          * The problem here is that sometimes the container_of() macro is itself
145          * inside a macro and get_macro() only returns the name of the outside
146          * macro.
147          */

149         /*
150          * This actually an expression statement assignment but smatch_flow
151          * pre-mangles it for us so we only get the last chunk:
152          * sk = (typeof(sk))((char *)__mptr - offsetof(...))
153          */

155         macro = get_macro_name(right->pos);
156         if (!macro)
157                 return;
158         if (right->type != EXPR_CAST)
159                 return;
160         right = strip_expr(right);
161         if (right->type != EXPR_BINOP || right->op != '-' ||
162             right->left->type != EXPR_CAST)
163                 return;
164         right = strip_expr(right->left);
165         if (right->type != EXPR_SYMBOL)
166                 return;
167         if (!right->symbol->ident ||
168             strcmp(right->symbol->ident->name, "__mptr") != 0)
169                 return;
170         set_extra_expr_mod(expr->left, alloc_estate_range(valid_ptr_min_sval, va
171 }

173 static int match_next_bit(struct expression *call, void *unused, struct range_li
174 {
175         struct expression *start_arg;
176         struct expression *size_arg;
177         struct symbol *type;
178         sval_t min, max, tmp;

180         size_arg = get_argument_from_call_expr(call->args, 1);
181         /* btw. there isn't a start_arg for find_first_bit() */
182         start_arg = get_argument_from_call_expr(call->args, 2);

184         type = get_type(call);
185         min = sval_type_val(type, 0);
186         max = sval_type_val(type, sizeof(long long) * 8);

188         if (get_implied_max(size_arg, &tmp) && tmp.uvalue < max.value)
189                 max = tmp;
190         if (start_arg && get_implied_min(start_arg, &tmp) && !sval_is_negative(t
191                 min = tmp;
192         if (sval_cmp(min, max) > 0)
```



```
193                 max = min;
194         min = sval_cast(type, min);
195         max = sval_cast(type, max);
196         *rl = alloc_rl(min, max);
197         return 1;
198 }

200 static int match_fls(struct expression *call, void *unused, struct range_list **
201 {
202         struct expression *arg;
203         struct range_list *arg_rl;
204         sval_t zero = {};
205         sval_t start, end, sval;

207         start.type = &int_ctype;
208         start.value = 0;
209         end.type = &int_ctype;
210         end.value = 32;

212         arg = get_argument_from_call_expr(call->args, 0);
213         if (!get_implied_rl(arg, &arg_rl))
214                 return 0;
215         if (rl_to_sval(arg_rl, &sval)) {
216                 int i;

218                 for (i = 63; i >= 0; i--) {
219                         if (sval.uvalue & 1ULL << i)
220                                 break;
221                 }
222                 sval.value = i + 1;
223                 *rl = alloc_rl(sval, sval);
224                 return 1;
225         }
226         zero.type = rl_type(arg_rl);
227         if (!rl_has_sval(arg_rl, zero))
228                 start.value = 1;
229         *rl = alloc_rl(start, end);
230         return 1;
231 }



235 static void find_module_init_exit(struct symbol_list *sym_list)
236 {
237         struct symbol *sym;
238         struct symbol *fn;
239         struct statement *stmt;
240         char *name;
241         int init;
242         int count;

244         /*
245          * This is more complicated because Sparse ignores the "alias"
246          * attribute.  I search backwards because module_init() is normally at
247          * the end of the file.
248          */
249         count = 0;
250         FOR_EACH_PTR_REVERSE(sym_list, sym) {
251                 if (sym->type != SYM_NODE)
252                         continue;
253                 if (!(sym->ctype.modifiers & MOD_STATIC))
254                         continue;
255                 fn = get_base_type(sym);
256                 if (!fn)
257                         continue;
258                 if (fn->type != SYM_FN)
```

```
259                     continue;
260             if (!sym->ident)
261                     continue;
262             if (!fn->inline_stmt)
263                     continue;
264             if (strcmp(sym->ident->name, "__inittest") == 0)
265                     init = 1;
266             else if (strcmp(sym->ident->name, "__exittest") == 0)
267                     init = 0;
268             else
269                     continue;

271             count++;

273             stmt = first_ptr_list((struct ptr_list *)fn->inline_stmt->stmts)
274             if (!stmt || stmt->type != STMT_RETURN)
275                     continue;
276             name = expr_to_var(stmt->ret_value);
277             if (!name)
278                     continue;
279             if (init)
280                     sql_insert_function_ptr(name, "(struct module)->init");
281             else
282                     sql_insert_function_ptr(name, "(struct module)->exit");
283             free_string(name);
284             if (count >= 2)
285                     return;
286     } END_FOR_EACH_PTR_REVERSE(sym);
287 }

289 static void match_end_file(struct symbol_list *sym_list)
290 {
291     struct symbol *sym;

293     /* find the last static symbol in the file */
294     FOR_EACH_PTR_REVERSE(sym_list, sym) {
295             if (!(sym->ctype.modifiers & MOD_STATIC))
296                     continue;
297             if (!sym->scope)
298                     continue;
299             find_module_init_exit(sym->scope->symbols);
300             return;
301     } END_FOR_EACH_PTR_REVERSE(sym);
302 }

304 static struct expression *get_val_expr(struct expression *expr)
305 {
306     struct symbol *sym, *val;

308     if (expr->type != EXPR_DEREF)
309             return NULL;
310     expr = expr->deref;
311     if (expr->type != EXPR_SYMBOL)
312             return NULL;
313     if (strcmp(expr->symbol_name->name, "__u") != 0)
314             return NULL;
315     sym = get_base_type(expr->symbol);
316     val = first_ptr_list((struct ptr_list *)sym->symbol_list);
317     if (!val || strcmp(val->ident->name, "__val") != 0)
318             return NULL;
319     return member_expression(expr, '.', val->ident);
320 }

322 static void match__write_once_size(const char *fn, struct expression *call,
323                                    void *unused)
324 {
```



```
325     struct expression *dest, *data, *assign;
326     struct range_list *rl;

328     dest = get_argument_from_call_expr(call->args, 0);
329     if (dest->type != EXPR_PREOP || dest->op != '&')
330             return;
331     dest = strip_expr(dest->unop);

333     data = get_argument_from_call_expr(call->args, 1);
334     data = get_val_expr(data);
335     if (!data)
336             return;
337     get_absolute_rl(data, &rl);
338     assign = assign_expression(dest, '=', data);

340     __in_fake_assign++;
341     __split_expr(assign);
342     __in_fake_assign--;
343 }

345 static void match__read_once_size(const char *fn, struct expression *call,
346                                   void *unused)
347 {
348     struct expression *dest, *data, *assign;
349     struct symbol *type, *val_sym;

351     /*
352      * We want to change:
353      *      __read_once_size_nocheck(&(x), __u.__c, sizeof(x));
354      * into a fake assignment:
355      *      __u.val = x;
356      *
357      */

359     data = get_argument_from_call_expr(call->args, 0);
360     if (data->type != EXPR_PREOP || data->op != '&')
361             return;
362     data = strip_parens(data->unop);

364     dest = get_argument_from_call_expr(call->args, 1);
365     if (dest->type != EXPR_DEREF || dest->op != '.')
366             return;
367     if (!dest->member || strcmp(dest->member->name, "__c") != 0)
368             return;
369     dest = dest->deref;
370     type = get_type(dest);
371     if (!type)
372             return;
373     val_sym = first_ptr_list((struct ptr_list *)type->symbol_list);
374     dest = member_expression(dest, '.', val_sym->ident);

376     assign = assign_expression(dest, '=', data);
377     __in_fake_assign++;
378     __split_expr(assign);
379     __in_fake_assign--;
380 }

382 void check_kernel(int id)
383 {
384     if (option_project != PROJ_KERNEL)
385             return;

387     add_implied_return_hook("ERR_PTR", &implied_err_cast_return, NULL);
388     add_implied_return_hook("ERR_CAST", &implied_err_cast_return, NULL);
389     add_implied_return_hook("PTR_ERR", &implied_err_cast_return, NULL);
390     add_hook(hack_ERR_PTR, AFTER_DEF_HOOK);
```

```
391         return_implies_state("IS_ERR_OR_NULL", 0, 0, &match_param_valid_ptr, (vo
392         return_implies_state("IS_ERR_OR_NULL", 1, 1, &match_param_err_or_null, (
393         return_implies_state("IS_ERR", 0, 0, &match_not_err, NULL);
394         return_implies_state("IS_ERR", 1, 1, &match_err, NULL);
395         return_implies_state("tomoyo_memory_ok", 1, 1, &match_param_valid_ptr, (

397         add_macro_assign_hook_extra("container_of", &match_container_of_macro, N
398         add_hook(match_container_of, ASSIGNMENT_HOOK);

400         add_implied_return_hook("find_next_bit", &match_next_bit, NULL);
401         add_implied_return_hook("find_next_zero_bit", &match_next_bit, NULL);
402         add_implied_return_hook("find_first_bit", &match_next_bit, NULL);
403         add_implied_return_hook("find_first_zero_bit", &match_next_bit, NULL);

405         add_implied_return_hook("fls", &match_fls, NULL);
406         add_implied_return_hook("fls64", &match_fls, NULL);

408         add_function_hook("__ftrace_bad_type", &__match_nullify_path_hook, NULL)
409         add_function_hook("__write_once_size", &match__write_once_size, NULL);

411         add_function_hook("__read_once_size", &match__read_once_size, NULL);
412         add_function_hook("__read_once_size_nocheck", &match__read_once_size, NU

414         if (option_info)
415                 add_hook(match_end_file, END_FILE_HOOK);
416 }
```

```
**********************************************************
   53938 Fri Dec 21 15:00:04 2018
new/usr/src/tools/smatch/src/check_kernel_printf.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Rasmus Villemoes.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <assert.h>
#include <ctype.h>
#include <string.h>
#include "smatch.h"
#include "smatch_slist.h"

#define spam(args...) do {                      \
        if (option_spammy)                      \
                sm_msg(args);                   \
        } while (0)

static int my_id;

/*
 * Much of this is taken directly from the kernel (mostly vsprintf.c),
 * with a few modifications here and there.
 */

#define KERN_SOH_ASCII  '\001'

typedef unsigned char u8;
typedef signed short s16;

#define SIGN    1               /* unsigned/signed, must be 1 */
#define LEFT    2               /* left justified */
#define PLUS    4               /* show plus */
#define SPACE   8               /* space if plus */
#define ZEROPAD 16              /* pad with zero, must be 16 == '0' - ' ' */
#define SMALL   32              /* use lowercase in hex (must be 32 == 0x20) */
#define SPECIAL 64              /* prefix hex with "0x", octal with "0" */

enum format_type {
        FORMAT_TYPE_NONE, /* Just a string part */
        FORMAT_TYPE_WIDTH,
        FORMAT_TYPE_PRECISION,
        FORMAT_TYPE_CHAR,
        FORMAT_TYPE_STR,
        FORMAT_TYPE_PTR,
        FORMAT_TYPE_PERCENT_CHAR,
        FORMAT_TYPE_INVALID,
        FORMAT_TYPE_LONG_LONG,
        FORMAT_TYPE_ULONG,
        FORMAT_TYPE_LONG,
```



```
        FORMAT_TYPE_UBYTE,
        FORMAT_TYPE_BYTE,
        FORMAT_TYPE_USHORT,
        FORMAT_TYPE_SHORT,
        FORMAT_TYPE_UINT,
        FORMAT_TYPE_INT,
        FORMAT_TYPE_SIZE_T,
        FORMAT_TYPE_PTRDIFF,
        FORMAT_TYPE_NRCHARS, /* Reintroduced for this checker */
        FORMAT_TYPE_FLOAT, /* for various floating point formatters */
};

struct printf_spec {
        unsigned int    type:8;         /* format_type enum */
        signed int      field_width:24; /* width of output field */
        unsigned int    flags:8;        /* flags to number() */
        unsigned int    base:8;         /* number base, 8, 10 or 16 only */
        signed int      precision:16;   /* # of digits/chars */
} __packed;
#define FIELD_WIDTH_MAX ((1 << 23) - 1)
#define PRECISION_MAX ((1 << 15) - 1)
extern char __check_printf_spec[1-2*(sizeof(struct printf_spec) != 8)];

static int
skip_atoi(const char **s)
{
        int i = 0;

        while (isdigit(**s))
                i = i*10 + *((*s)++) - '0';

        return i;
}

static int
format_decode(const char *fmt, struct printf_spec *spec)
{
        const char *start = fmt;
        char qualifier;

        /* we finished early by reading the field width */
        if (spec->type == FORMAT_TYPE_WIDTH) {
                if (spec->field_width < 0) {
                        spec->field_width = -spec->field_width;
                        spec->flags |= LEFT;
                }
                spec->type = FORMAT_TYPE_NONE;
                goto precision;
        }

        /* we finished early by reading the precision */
        if (spec->type == FORMAT_TYPE_PRECISION) {
                if (spec->precision < 0)
                        spec->precision = 0;

                spec->type = FORMAT_TYPE_NONE;
                goto qualifier;
        }

        /* By default */
        spec->type = FORMAT_TYPE_NONE;

        for (; *fmt ; ++fmt) {
                if (*fmt == '%')
                        break;
        }
```

```
128         /* Return the current non-format string */
129         if (fmt != start || !*fmt)
130                 return fmt - start;
131
132         /* Process flags */
133         spec->flags = 0;
134
135         while (1) { /* this also skips first '%' */
136                 bool found = true;
137
138                 ++fmt;
139
140                 switch (*fmt) {
141                 case '-': spec->flags |= LEFT;    break;
142                 case '+': spec->flags |= PLUS;    break;
143                 case ' ': spec->flags |= SPACE;   break;
144                 case '#': spec->flags |= SPECIAL; break;
145                 case '0': spec->flags |= ZEROPAD; break;
146                 default:  found = false;
147                 }
148
149                 if (!found)
150                         break;
151         }
152
153         /* get field width */
154         spec->field_width = -1;
155
156         if (isdigit(*fmt))
157                 spec->field_width = skip_atoi(&fmt);
158         else if (*fmt == '*') {
159                 /* it's the next argument */
160                 spec->type = FORMAT_TYPE_WIDTH;
161                 return ++fmt - start;
162         }
163
164 precision:
165         /* get the precision */
166         spec->precision = -1;
167         if (*fmt == '.') {
168                 ++fmt;
169                 if (isdigit(*fmt)) {
170                         spec->precision = skip_atoi(&fmt);
171                         if (spec->precision < 0)
172                                 spec->precision = 0;
173                 } else if (*fmt == '*') {
174                         /* it's the next argument */
175                         spec->type = FORMAT_TYPE_PRECISION;
176                         return ++fmt - start;
177                 }
178         }
179
180 qualifier:
181         /* get the conversion qualifier */
182         qualifier = 0;
183         if (*fmt == 'h' || _tolower(*fmt) == 'l' ||
184             _tolower(*fmt) == 'z' || *fmt == 't') {
185                 qualifier = *fmt++;
186                 if (qualifier == *fmt) {
187                         if (qualifier == 'l') {
188                                 qualifier = 'L';
189                                 ++fmt;
190                         } else if (qualifier == 'h') {
191                                 qualifier = 'H';
192                                 ++fmt;
```



```
193                         } else {
194                                 sm_warning("invalid repeated qualifier '%c'", *f
195                         }
196                 }
197         }
198
199         /* default base */
200         spec->base = 10;
201         switch (*fmt) {
202         case 'c':
203                 if (qualifier)
204                         sm_warning("qualifier '%c' ignored for %%c specifier", q
205
206                 spec->type = FORMAT_TYPE_CHAR;
207                 return ++fmt - start;
208
209         case 's':
210                 if (qualifier)
211                         sm_warning("qualifier '%c' ignored for %%s specifier", q
212
213                 spec->type = FORMAT_TYPE_STR;
214                 return ++fmt - start;
215
216         case 'p':
217                 spec->type = FORMAT_TYPE_PTR;
218                 return ++fmt - start;
219
220         case '%':
221                 spec->type = FORMAT_TYPE_PERCENT_CHAR;
222                 return ++fmt - start;
223
224         /* integer number formats - set up the flags and "break" */
225         case 'o':
226                 spec->base = 8;
227                 break;
228
229         case 'x':
230                 spec->flags |= SMALL;
231
232         case 'X':
233                 spec->base = 16;
234                 break;
235
236         case 'd':
237         case 'i':
238                 spec->flags |= SIGN;
239         case 'u':
240                 break;
241
242         case 'n':
243                 spec->type = FORMAT_TYPE_NRCHARS;
244                 return ++fmt - start;
245
246         case 'a': case 'A':
247         case 'e': case 'E':
248         case 'f': case 'F':
249         case 'g': case 'G':
250                 spec->type = FORMAT_TYPE_FLOAT;
251                 return ++fmt - start;
252
253         default:
254                 spec->type = FORMAT_TYPE_INVALID;
255                 /* Unlike the kernel code, we 'consume' the invalid
256                  * character so that it can get included in the
257                  * report. After that, we bail out. */
258                 return ++fmt - start;
```

```
259         }

261         if (qualifier == 'L')
262                 spec->type = FORMAT_TYPE_LONG_LONG;
263         else if (qualifier == 'l') {
264                 if (spec->flags & SIGN)
265                         spec->type = FORMAT_TYPE_LONG;
266                 else
267                         spec->type = FORMAT_TYPE_ULONG;
268         } else if (_tolower(qualifier) == 'z') {
269                 spec->type = FORMAT_TYPE_SIZE_T;
270         } else if (qualifier == 't') {
271                 spec->type = FORMAT_TYPE_PTRDIFF;
272         } else if (qualifier == 'H') {
273                 if (spec->flags & SIGN)
274                         spec->type = FORMAT_TYPE_BYTE;
275                 else
276                         spec->type = FORMAT_TYPE_UBYTE;
277         } else if (qualifier == 'h') {
278                 if (spec->flags & SIGN)
279                         spec->type = FORMAT_TYPE_SHORT;
280                 else
281                         spec->type = FORMAT_TYPE_USHORT;
282         } else {
283                 if (spec->flags & SIGN)
284                         spec->type = FORMAT_TYPE_INT;
285                 else
286                         spec->type = FORMAT_TYPE_UINT;
287         }

289         return ++fmt - start;
290 }

292 static int is_struct_tag(struct symbol *type, const char *tag)
293 {
294         return type->type == SYM_STRUCT && type->ident && !strcmp(type->ident->n
295 }

297 static int has_struct_tag(struct symbol *type, const char *tag)
298 {
299         struct symbol *tmp;

301         if (type->type == SYM_STRUCT)
302                 return is_struct_tag(type, tag);
303         if (type->type == SYM_UNION) {
304                 FOR_EACH_PTR(type->symbol_list, tmp) {
305                         tmp = get_real_base_type(tmp);
306                         if (tmp && is_struct_tag(tmp, tag))
307                                 return 1;
308                 } END_FOR_EACH_PTR(tmp);
309         }
310         return 0;
311 }

313 static int is_char_type(struct symbol *type)
314 {
315         return type == &uchar_ctype || type == &char_ctype || type == &schar_cty
316 }

318 /*
319  * I have absolutely no idea if this is how one is supposed to get the
320  * symbol representing a typedef, but it seems to work.
321  */
322 struct typedef_lookup {
323         const char *name;
324         struct symbol *sym;
```



```
325         int failed;
326 };

328 static struct symbol *_typedef_lookup(const char *name)
329 {
330         struct ident *id;
331         struct symbol *node;

333         id = built_in_ident(name);
334         if (!id)
335                 return NULL;
336         node = lookup_symbol(id, NS_TYPEDEF);
337         if (!node || node->type != SYM_NODE)
338                 return NULL;
339         return get_real_base_type(node);
340 }

342 static void typedef_lookup(struct typedef_lookup *tl)
343 {
344         if (tl->sym || tl->failed)
345                 return;
346         tl->sym = _typedef_lookup(tl->name);
347         if (!tl->sym) {
348                 sm_perror(" could not find typedef '%s'", tl->name);
349                 tl->failed = 1;
350         }
351 }


354 static void ip4(const char *fmt, struct symbol *type, struct symbol *basetype, i
355 {
356         enum { ENDIAN_BIG, ENDIAN_LITTLE, ENDIAN_HOST } endian = ENDIAN_BIG;

358         assert(fmt[0] == 'i' || fmt[0] == 'I');
359         assert(fmt[1] == '4');

361         if (isalnum(fmt[2])) {
362                 switch (fmt[2]) {
363                 case 'h':
364                         endian = ENDIAN_HOST;
365                         break;
366                 case 'l':
367                         endian = ENDIAN_LITTLE;
368                         break;
369                 case 'n':
370                 case 'b':
371                         endian = ENDIAN_BIG;
372                         break;
373                 default:
374                         sm_warning("'%%p%c4' can only be followed by one of [hnb
375                 }
376                 if (isalnum(fmt[3]))
377                         sm_warning("'%%p%c4' can only be followed by precisely o
378         }


381         if (type->ctype.modifiers & MOD_NODEREF)
382                 sm_error("passing __user pointer to '%%p%c4'", fmt[0]);

384         /*
385          * If we have a pointer to char/u8/s8, we expect the caller to
386          * handle endianness; I don't think there's anything we can
387          * do. I'd like to check that if we're passed a pointer to a
388          * __bitwise u32 (most likely a __be32), we should have endian
389          * == ENDIAN_BIG. But I can't figure out how to get that
390          * information (it also seems to require ensuring certain
```

```
391          * macros are defined). But struct in_addr certainly consists
392          * of only a single __be32, so in that case we can do a check.
393          */
394         if (is_char_type(basetype))
395                 return;

397         if (is_struct_tag(basetype, "in_addr") && endian != ENDIAN_BIG)
398                 sm_warning("passing struct in_addr* to '%%p%c4%c', is the endian

400         /* ... */
401 }

403 static void ip6(const char *fmt, struct symbol *type, struct symbol *basetype, i
404 {
405         assert(fmt[0] == 'i' || fmt[0] == 'I');
406         assert(fmt[1] == '6');

408         if (isalnum(fmt[2])) {
409                 if (fmt[2] != 'c')
410                         sm_warning("'%%p%c6' can only be followed by c", fmt[0])
411                 else if (fmt[0] == 'i')
412                         sm_warning("'%%pi6' does not allow flag c");
413                 if (isalnum(fmt[3]))
414                         sm_warning("'%%p%c6%c' cannot be followed by other alpha
415         }

417         if (type->ctype.modifiers & MOD_NODEREF)
418                 sm_error("passing __user pointer to '%%p%c6'", fmt[0]);
419 }

421 static void ipS(const char *fmt, struct symbol *type, struct symbol *basetype, i
422 {
423         const char *f;

425         assert(tolower(fmt[0]) == 'i');
426         assert(fmt[1] == 'S');

428         for (f = fmt+2; isalnum(*f); ++f) {
429                 /* It's probably too anal checking for duplicate flags. */
430                 if (!strchr("pfschnbl", *f))
431                         sm_warning("'%%p%cS' cannot be followed by '%c'", fmt[0]
432         }

434         /*
435          * XXX: Should we also allow passing a pointer to a union, one
436          * member of which is a struct sockaddr? It may be slightly
437          * cleaner actually passing &u.raw instead of just &u, though
438          * the generated code is of course exactly the same. For now,
439          * we do accept struct sockaddr_in and struct sockaddr_in6,
440          * since those are easy to handle and rather harmless.
441          */
442         if (!has_struct_tag(basetype, "sockaddr") &&
443             !has_struct_tag(basetype, "sockaddr_in") &&
444             !has_struct_tag(basetype, "sockaddr_in6") &&
445             !has_struct_tag(basetype, "__kernel_sockaddr_storage"))
446                 sm_error("'%%p%cS' expects argument of type struct sockaddr *, "
447                         "argument %d has type '%s'", fmt[0], vaidx, type_to_str(
448 }

450 static void hex_string(const char *fmt, struct symbol *type, struct symbol *base
451 {
452         assert(fmt[0] == 'h');
453         if (isalnum(fmt[1])) {
454                 if (!strchr("CDN", fmt[1]))
455                         sm_warning("'%%ph' cannot be followed by '%c'", fmt[1]);
456                 if (isalnum(fmt[2]))
```



```
457                         sm_warning("'%%ph' can be followed by at most one of [CD
458         }
459         if (type->ctype.modifiers & MOD_NODEREF)
460                 sm_error("passing __user pointer to %%ph");
461 }

463 static void escaped_string(const char *fmt, struct symbol *type, struct symbol *
464 {
465         assert(fmt[0] == 'E');
466         while (isalnum(*++fmt)) {
467                 if (!strchr("achnops", *fmt))
468                         sm_warning("%%pE can only be followed by a combination o
469         }
470         if (type->ctype.modifiers & MOD_NODEREF)
471                 sm_error("passing __user pointer to %%pE");
472 }

474 static void resource_string(const char *fmt, struct symbol *type, struct symbol
475 {
476         assert(tolower(fmt[0]) == 'r');
477         if (!is_struct_tag(basetype, "resource")) {
478                 sm_error("'%%p%c' expects argument of type struct resource *, "
479                         "but argument %d has type '%s'", fmt[0], vaidx, type_to_
480         }
481         if (isalnum(fmt[1]))
482                 sm_warning("'%%p%c' cannot be followed by '%c'", fmt[0], fmt[1])
483 }

485 static void mac_address_string(const char *fmt, struct symbol *type, struct symb
486 {
487         assert(tolower(fmt[0]) == 'm');
488         if (isalnum(fmt[1])) {
489                 if (!(fmt[1] == 'F' || fmt[1] == 'R'))
490                         sm_warning("'%%p%c' cannot be followed by '%c'", fmt[0],
491                 if (fmt[0] == 'm' && fmt[1] == 'F')
492                         sm_warning("it is pointless to pass flag F to %%pm");
493                 if (isalnum(fmt[2]))
494                         sm_warning("'%%p%c%c' cannot be followed by other alphan
495         }
496         /* Technically, bdaddr_t is a typedef for an anonymous struct, but this
497         if (!is_char_type(basetype) && !is_struct_tag(basetype, "bdaddr_t") && b
498                 sm_warning("'%%p%c' expects argument of type u8 * or bdaddr_t *,
499                         fmt[0], vaidx, type_to_str(type));
500         }
501         if (type->ctype.modifiers & MOD_NODEREF)
502                 sm_error("passing __user pointer to '%%p%c'", fmt[0]);
503 }

505 static void dentry_file(const char *fmt, struct symbol *type, struct symbol *bas
506 {
507         const char *tag;

509         assert(tolower(fmt[0]) == 'd');
510         tag = fmt[0] == 'd' ? "dentry" : "file";

512         if (isalnum(fmt[1])) {
513                 if (!strchr("234", fmt[1]))
514                         sm_warning("'%%p%c' can only be followed by one of [234]
515                 if (isalnum(fmt[2]))
516                         sm_warning("'%%p%c%c' cannot be followed by '%c'", fmt[0
517         }

519         if (!is_struct_tag(basetype, tag))
520                 sm_error("'%%p%c' expects argument of type struct '%s*', argumen
521                         fmt[0], tag, vaidx, type_to_str(type));
522 }
```

```
524 static void check_clock(const char *fmt, struct symbol *type, struct symbol *bas
525 {
526         assert(fmt[0] == 'C');
527         if (isalnum(fmt[1])) {
528                 if (!strchr("nr", fmt[1]))
529                         sm_warning("'%%pC' can only be followed by one of [nr]")
530                 if (isalnum(fmt[2]))
531                         sm_warning("'%%pC%c' cannot be followed by '%c'", fmt[1]
532         }
533         if (!is_struct_tag(basetype, "clk"))
534                 sm_error("'%%pC' expects argument of type 'struct clk*', argumen
535                        vaidx, type_to_str(type));
536 }

538 static void va_format(const char *fmt, struct symbol *type, struct symbol *baset
539 {
540         assert(fmt[0] == 'V');
541         if (isalnum(fmt[1]))
542                 sm_warning("%%pV cannot be followed by any alphanumerics");
543         if (!is_struct_tag(basetype, "va_format"))
544                 sm_error("%%pV expects argument of type struct va_format*, argum
545 }

547 static void netdev_feature(const char *fmt, struct symbol *type, struct symbol *
548 {
549         static struct typedef_lookup netdev = { .name = "netdev_features_t" };

551         assert(fmt[0] == 'N');
552         if (fmt[1] != 'F') {
553                 sm_error("%%pN must be followed by 'F'");
554                 return;
555         }
556         if (isalnum(fmt[2]))
557                 sm_warning("%%pNF cannot be followed by '%c'", fmt[2]);

559         typedef_lookup(&netdev);
560         if (!netdev.sym)
561                 return;
562         if (basetype != netdev.sym)
563                 sm_error("%%pNF expects argument of type netdev_features_t*, arg
564                         vaidx, type_to_str(type));

566 }
567 static void address_val(const char *fmt, struct symbol *type, struct symbol *bas
568 {
569         static struct typedef_lookup dma = { .name = "dma_addr_t" };
570         static struct typedef_lookup phys = { .name = "phys_addr_t" };
571         struct typedef_lookup *which = &phys;
572         const char *suf = "";
573         assert(fmt[0] == 'a');

575         if (isalnum(fmt[1])) {
576                 switch (fmt[1]) {
577                 case 'd':
578                         which = &dma;
579                         suf = "d";
580                         break;
581                 case 'p':
582                         suf = "p";
583                         break;
584                 default:
585                         sm_error("'%%pa' can only be followed by one of [dp]");
586                 }
587                 if (isalnum(fmt[2]))
588                         sm_error("'%%pa%c' cannot be followed by '%c'", fmt[1],
```



```
589         }

591         typedef_lookup(which);
592         if (!which->sym)
593                 return;
594         if (basetype != which->sym) {
595                 sm_error("'%%pa%s' expects argument of type '%s*', argument %d h
596                         suf, which->name, vaidx, type_to_str(type));
597         }
598 }

600 static void block_device(const char *fmt, struct symbol *type, struct symbol *ba
601 {
602         const char *tag = "block_device";

604         assert(fmt[0] == 'g');
605         if (isalnum(fmt[1])) {
606                 sm_warning("%%pg cannot be followed by '%c'", fmt[1]);
607         }
608         if (!is_struct_tag(basetype, tag))
609                 sm_error("'%%p%c' expects argument of type struct '%s*', argumen
610                         fmt[0], tag, vaidx, type_to_str(type));
611 }

613 static void flag_string(const char *fmt, struct symbol *type, struct symbol *bas
614 {
615         static struct typedef_lookup gfp = { .name = "gfp_t" };

617         assert(fmt[0] == 'G');
618         if (!isalnum(fmt[1])) {
619                 sm_error("%%pG must be followed by one of [gpv]");
620                 return;
621         }
622         switch (fmt[1]) {
623         case 'p':
624         case 'v':
625                 if (basetype != &ulong_ctype)
626                         sm_error("'%%pG%c' expects argument of type 'unsigned lo
627                                 fmt[1], vaidx, type_to_str(type));
628                 break;
629         case 'g':
630                 typedef_lookup(&gfp);
631                 if (basetype != gfp.sym)
632                         sm_error("'%%pGg' expects argument of type 'gfp_t *', ar
633                                 vaidx, type_to_str(type));
634                 break;
635         default:
636                 sm_error("'%%pG' must be followed by one of [gpv]");
637         }
638 }

640 static void device_node_string(const char *fmt, struct symbol *type, struct symb
641 {
642         if (fmt[1] != 'F') {
643                 sm_error("%%pO can only be followed by 'F'");
644                 return;
645         }
646         if (!is_struct_tag(basetype, "device_node"))
647                 sm_error("'%%pOF' expects argument of type 'struct device_node*'
648                        vaidx, type_to_str(type));
649 }

651 static void
652 pointer(const char *fmt, struct expression *arg, int vaidx)
653 {
654         struct symbol *type, *basetype;
```

```
656         type = get_type(arg);
657         if (!type) {
658                 sm_warning("could not determine type of argument %d", vaidx);
659                 return;
660         }
661         if (!is_ptr_type(type)) {
662                 sm_error("%%p expects pointer argument, but argument %d has type
663                         vaidx, type_to_str(type));
664                 return;
665         }
666         /* Just plain %p, nothing to check. */
667         if (!isalnum(*fmt))
668                 return;

670         basetype = get_real_base_type(type);
671         if (is_void_type(basetype))
672                 return;
673         /*
674          * Passing a pointer-to-array is harmless, but most likely one
675          * meant to pass pointer-to-first-element. If basetype is
676          * array type, we issue a notice and "dereference" the types
677          * once more.
678          */
679         if (basetype->type == SYM_ARRAY) {
680                 spam("note: passing pointer-to-array; is the address-of redundan
681                 type = basetype;
682                 basetype = get_real_base_type(type);
683         }

685         /*
686          * We pass both the type and the basetype to the helpers. If,
687          * for example, the pointer is really a decayed array which is
688          * passed to %pI4, we might want to check that it is in fact
689          * an array of four bytes. But most are probably only
690          * interested in whether the basetype makes sense. Also, the
691          * pointer may carry some annotation such as __user which
692          * might be worth checking in the handlers which actually
693          * dereference the pointer.
694          */

696         switch (*fmt) {
697         case 'b':
698         case 'F':
699         case 'f':
700         case 'S':
701         case 's':
702         case 'B':
703                 /* Can we do anything sensible? Check that the arg is a function
704                 break;

706         case 'R':
707         case 'r':
708                 resource_string(fmt, type, basetype, vaidx);
709                 break;
710         case 'M':
711         case 'm':
712                 mac_address_string(fmt, type, basetype, vaidx);
713                 break;
714         case 'I':
715         case 'i':
716                 switch (fmt[1]) {
717                 case '4':
718                         ip4(fmt, type, basetype, vaidx);
719                         break;
720                 case '6':
```



```
721                         ip6(fmt, type, basetype, vaidx);
722                         break;
723                 case 'S':
724                         ipS(fmt, type, basetype, vaidx);
725                         break;
726                 default:
727                         sm_warning("'%%p%c' must be followed by one of [46S]", f
728                         break;
729                 }
730                 break;
731         /*
732          * %pE and %ph can handle any valid pointer. We still check
733          * whether all the subsequent alphanumerics are valid for the
734          * particular %pX conversion.
735          */
736         case 'E':
737                 escaped_string(fmt, type, basetype, vaidx);
738                 break;
739         case 'h':
740                 hex_string(fmt, type, basetype, vaidx);
741                 break;
742         case 'U': /* TODO */
743                 break;
744         case 'V':
745                 va_format(fmt, type, basetype, vaidx);
746                 break;
747         case 'K': /* TODO */
748                 break;
749         case 'N':
750                 netdev_feature(fmt, type, basetype, vaidx);
751                 break;
752         case 'a':
753                 address_val(fmt, type, basetype, vaidx);
754                 break;
755         case 'D':
756         case 'd':
757                 dentry_file(fmt, type, basetype, vaidx);
758                 break;
759         case 'C':
760                 check_clock(fmt, type, basetype, vaidx);
761                 break;
762         case 'g':
763                 block_device(fmt, type, basetype, vaidx);
764                 break;
765         case 'G':
766                 flag_string(fmt, type, basetype, vaidx);
767                 break;
768         case 'O':
769                 device_node_string(fmt, type, basetype, vaidx);
770                 break;
771         case 'x':
772                 /* 'x' is for an unhashed pointer */
773                 break;
774         default:
775                 sm_error("unrecognized %%p extension '%c', treated as normal %%p
776         }
777 }

779 /*
780  * A common error is to pass a "char" or "signed char" to %02x (or
781  * %.2X or some other variant). This can actually be a security
782  * problem, because a lot of code expects this to produce exactly two
783  * characters of output. Unfortunately this also produces false
784  * positives, since we're sometimes in arch-specific code on an arch
785  * where char is always unsigned.
786  */
```

```
787 static void
788 hexbyte(const char *fmt, int fmt_len, struct expression *arg, int vaidx, struct
789 {
790         struct symbol *type;

792         /*
793          * For now, just check the most common and obvious, which is
794          * roughly %[.0]2[xX].
795          */
796         if (spec.field_width != 2 && spec.precision != 2)
797                 return;
798         if (spec.base != 16)
799                 return;

801         type = get_type(arg);
802         if (!type) {
803                 sm_warning("could not determine type of argument %d", vaidx);
804                 return;
805         }
806         if (type == &char_ctype || type == &schar_ctype)
807                 sm_warning("argument %d to %.*s specifier has type '%s'",
808                        vaidx, fmt_len, fmt, type_to_str(type));
809 }

811 static int
812 check_format_string(const char *fmt, const char *caller)
813 {
814         const char *f;

816         for (f = fmt; *f; ++f) {
817                 unsigned char c = *f;
818                 switch (c) {
819                 case KERN_SOH_ASCII:
820                         /*
821                          * This typically arises from bad conversion
822                          * to pr_*, e.g. pr_warn(KERN_WARNING "something").
823                          */
824                         if (f != fmt)
825                                 sm_warning("KERN_* level not at start of string"
826                         /*
827                          * In a very few cases, the level is actually
828                          * computed and passed via %c, as in KERN_SOH
829                          * "%c...". printk explicitly supports
830                          * this.
831                          */
832                         if (!(('0' <= f[1] && f[1] <= '7') ||
833                               f[1] == 'd' || /* KERN_DEFAULT */
834                               f[1] == 'c' || /* KERN_CONT */
835                               (f[1] == '%' && f[2] == 'c')))
836                                 sm_warning("invalid KERN_* level: KERN_SOH_ASCII
837                         break;
838                 case '\t':
839                 case '\n':
840                 case '\r':
841                 case 0x20 ... 0x7e:
842                         break;
843                 case 0x80 ... 0xff:
844                         sm_warning("format string contains non-ascii character '
845                         break;
846                 case 0x08:
847                         if (f == fmt)
848                                 break;
849                         /* fall through */
850                 default:
851                         sm_warning("format string contains unusual character '\\
852                         break;
```



```
853                 }
854         }

856         f = strstr(fmt, caller);
857         if (f && strstr(f+1, caller))
858                 sm_warning("format string contains name of enclosing function '%

860         return f != NULL;
861 }

863 static int arg_is___func__(struct expression *arg)
864 {
865         if (arg->type != EXPR_SYMBOL)
866                 return 0;
867         return !strcmp(arg->symbol_name->name, "__func__") ||
868                !strcmp(arg->symbol_name->name, "__FUNCTION__") ||
869                !strcmp(arg->symbol_name->name, "__PRETTY_FUNCTION__");
870 }
871 static int arg_contains_caller(struct expression *arg, const char *caller)
872 {
873         if (arg->type != EXPR_STRING)
874                 return 0;
875         return strstr(arg->string->data, caller) != NULL;
876 }

878 static int is_array_of_const_char(struct symbol *sym)
879 {
880         struct symbol *base = sym->ctype.base_type;
881         if (base->type != SYM_ARRAY)
882                 return 0;
883         if (!(base->ctype.modifiers & MOD_CONST))
884                 return 0;
885         if (!is_char_type(base->ctype.base_type)) {
886                 spam("weird: format argument is array of const '%s'", type_to_st
887                 return 0;
888         }
889         return 1;
890 }

892 static int is_const_pointer_to_const_char(struct symbol *sym)
893 {
894         struct symbol *base = sym->ctype.base_type;
895         if (!(sym->ctype.modifiers & MOD_CONST))
896                 return 0;
897         if (base->type != SYM_PTR)
898                 return 0;
899         if (!(base->ctype.modifiers & MOD_CONST))
900                 return 0;
901         if (!is_char_type(base->ctype.base_type)) {
902                 spam("weird: format argument is pointer to const '%s'", type_to_
903                 return 0;
904         }
905         return 1;
906 }

908 static int unknown_format(struct expression *expr)
909 {
910         struct state_list *slist;

912         slist = get_strings(expr);
913         if (!slist)
914                 return 1;
915         if (slist_has_state(slist, &undefined))
916                 return 1;
917         free_slist(&slist);
918         return 0;
```

```
}

static bool has_hex_prefix(const char *orig_fmt, const char *old_fmt)
{
        return old_fmt >= orig_fmt + 2 &&
                old_fmt[-2] == '0' && _tolower(old_fmt[-1]) == 'x';
}

static bool is_integer_specifier(int type)
{
        switch (type) {
        case FORMAT_TYPE_LONG_LONG:
        case FORMAT_TYPE_ULONG:
        case FORMAT_TYPE_LONG:
        case FORMAT_TYPE_UBYTE:
        case FORMAT_TYPE_BYTE:
        case FORMAT_TYPE_USHORT:
        case FORMAT_TYPE_SHORT:
        case FORMAT_TYPE_UINT:
        case FORMAT_TYPE_INT:
        case FORMAT_TYPE_SIZE_T:
        case FORMAT_TYPE_PTRDIFF:
                return true;
        default:
                return false;
        }
}

static int
is_cast_expr(struct expression *expr)
{
        if (!expr)
                return 0;

        switch (expr->type) {
        case EXPR_CAST:
        case EXPR_FORCE_CAST:
                /* not EXPR_IMPLIED_CAST for our purposes */
                return 1;
        default:
                return 0;
        }
}

static void
check_cast_from_pointer(const char *fmt, int len, struct expression *arg, int va
{
        /*
         * This can easily be fooled by passing 0+(long)ptr or doing
         * "long local_var = (long)ptr" and passing local_var to
         * %lx. Tough.
         */
        if (!is_cast_expr(arg))
                return;
        while (is_cast_expr(arg))
                arg = arg->cast_expression;
        if (is_ptr_type(get_final_type(arg)))
                sm_warning("argument %d to %.*s specifier is cast from pointer",
                        va_idx, len, fmt);
}

static void
do_check_printf_call(const char *caller, const char *name, struct expression *ca
{
        struct printf_spec spec = {0};
        const char *fmt, *orig_fmt;
```



```
        int caller_in_fmt;

        fmtexpr = strip_parens(fmtexpr);
        if (fmtexpr->type == EXPR_CONDITIONAL) {
                do_check_printf_call(caller, name, callexpr, fmtexpr->cond_true
                do_check_printf_call(caller, name, callexpr, fmtexpr->cond_false
                return;
        }
        if (fmtexpr->type == EXPR_SYMBOL) {
                /*
                 * If the symbol has an initializer, we can handle
                 *
                 *   const char foo[] = "abc";         and
                 *   const char * const foo = "abc";
                 *
                 * We simply replace fmtexpr with the initializer
                 * expression. If foo is not one of the above, or if
                 * the initializer expression is somehow not a string
                 * literal, fmtexpr->type != EXPR_STRING will trigger
                 * below and we'll spam+return.
                 */
                struct symbol *sym = fmtexpr->symbol;
                if (sym && sym->initializer &&
                    (is_array_of_const_char(sym) ||
                     is_const_pointer_to_const_char(sym))) {
                        fmtexpr = strip_parens(sym->initializer);
                }
        }

        if (fmtexpr->type != EXPR_STRING) {
                if (!unknown_format(fmtexpr))
                        return;
                /*
                 * Since we're now handling both ?: and static const
                 * char[] arguments, we don't get as much noise. It's
                 * still spammy, though.
                 */
                spam("warn: call of '%s' with non-constant format argument", nam
                return;
        }

        orig_fmt = fmt = fmtexpr->string->data;
        caller_in_fmt = check_format_string(fmt, caller);

        while (*fmt) {
                const char *old_fmt = fmt;
                int read = format_decode(fmt, &spec);
                struct expression *arg;

                fmt += read;
                if (spec.type == FORMAT_TYPE_NONE ||
                    spec.type == FORMAT_TYPE_PERCENT_CHAR)
                        continue;

                /*
                 * vaidx is currently the correct 0-based index for
                 * get_argument_from_call_expr. We post-increment it
                 * here so that it is the correct 1-based index for
                 * all the handlers below. This of course requires
                 * that we handle all FORMAT_TYPE_* things not taking
                 * an argument above.
                 */
                arg = get_argument_from_call_expr(callexpr->args, vaidx++);

                if (spec.flags & SPECIAL && has_hex_prefix(orig_fmt, old_fmt))
                        sm_warning("'%.2s' prefix is redundant when # flag is us
```

```
1051                if (is_integer_specifier(spec.type)) {
1052                        if (spec.base != 16 && has_hex_prefix(orig_fmt, old_fmt)
1053                                sm_warning("'%.2s' prefix is confusing together
1054                                        old_fmt-2, (int)(fmt-old_fmt), old_fmt);
1055
1056                        check_cast_from_pointer(old_fmt, read, arg, vaidx);
1057                }
1058
1059                switch (spec.type) {
1060                /* case FORMAT_TYPE_NONE: */
1061                /* case FORMAT_TYPE_PERCENT_CHAR: */
1062                /*      break; */
1063
1064                case FORMAT_TYPE_INVALID:
1065                        sm_error("format specifier '%.*s' invalid", read, old_fm
1066                        return;
1067
1068                case FORMAT_TYPE_FLOAT:
1069                        sm_error("no floats in the kernel; invalid format specif
1070                        return;
1071
1072                case FORMAT_TYPE_NRCHARS:
1073                        sm_error("%%n not supported in kernel");
1074                        return;
1075
1076                case FORMAT_TYPE_WIDTH:
1077                case FORMAT_TYPE_PRECISION:
1078                        /* check int argument */
1079                        break;
1080
1081                case FORMAT_TYPE_STR:
1082                        /*
1083                         * If the format string already contains the
1084                         * function name, it probably doesn't make
1085                         * sense to pass __func__ as well (or rather
1086                         * vice versa: If pr_fmt(fmt) has been defined
1087                         * to '"%s: " fmt, __func__', it doesn't make
1088                         * sense to use a format string containing the
1089                         * function name).
1090                         *
1091                         * This produces a lot of hits. They are not
1092                         * false positives, but it is easier to handle
1093                         * the things which don't occur that often
1094                         * first, so we use spam().
1095                         */
1096                        if (caller_in_fmt) {
1097                                if (arg_is___func__(arg))
1098                                        spam("warn: passing __func__ while the f
1099                                                caller);
1100                                else if (arg_contains_caller(arg, caller))
1101                                        sm_warning("passing string constant '%s'
1102                                                arg->string->data, caller);
1103                        }
1104                        break;
1105
1106                case FORMAT_TYPE_PTR:
1107                        /* This is the most important part: Checking %p extensio
1108                        pointer(fmt, arg, vaidx);
1109                        while (isalnum(*fmt))
1110                                fmt++;
1111                        break;
1112
1113                case FORMAT_TYPE_CHAR:
1114
1115                case FORMAT_TYPE_UBYTE:
1116                case FORMAT_TYPE_BYTE:
```



```
1117                case FORMAT_TYPE_USHORT:
1118                case FORMAT_TYPE_SHORT:
1119                case FORMAT_TYPE_INT:
1120                        /* argument should have integer type of width <= sizeof(
1121                        break;
1122
1123                case FORMAT_TYPE_UINT:
1124                        hexbyte(old_fmt, fmt-old_fmt, arg, vaidx, spec);
1125                case FORMAT_TYPE_LONG:
1126                case FORMAT_TYPE_ULONG:
1127                case FORMAT_TYPE_LONG_LONG:
1128                case FORMAT_TYPE_PTRDIFF:
1129                case FORMAT_TYPE_SIZE_T:
1130                        break;
1131                }
1132
1133
1134        }
1135
1136        if (get_argument_from_call_expr(callexpr->args, vaidx))
1137                sm_warning("excess argument passed to '%s'", name);
1138
1139
1140 }
1141
1142 static void
1143 check_printf_call(const char *name, struct expression *callexpr, void *_info)
1144 {
1145         /*
1146          * Note: attribute(printf) uses 1-based indexing, but
1147          * get_argument_from_call_expr() uses 0-based indexing.
1148          */
1149         int info = PTR_INT(_info);
1150         int fmtidx = (info & 0xff) - 1;
1151         int vaidx = ((info >> 8) & 0xff) - 1;
1152         struct expression *fmtexpr;
1153         const char *caller = get_function();
1154
1155         if (!caller)
1156                 return;
1157
1158         /*
1159          * Calling a v*printf function with a literal format arg is
1160          * extremely rare, so we don't bother doing the only checking
1161          * we could do, namely checking that the format string is
1162          * valid.
1163          */
1164         if (vaidx < 0)
1165                 return;
1166
1167         /*
1168          * For the things we use the name of the calling function for,
1169          * it is more appropriate to skip a potential SyS_ prefix; the
1170          * same goes for leading underscores.
1171          */
1172         if (!strncmp(caller, "SyS_", 4))
1173                 caller += 4;
1174         while (*caller == '_')
1175                 ++caller;
1176
1177         /* Lack of format argument is a bug. */
1178         fmtexpr = get_argument_from_call_expr(callexpr->args, fmtidx);
1179         if (!fmtexpr) {
1180                 sm_error("call of '%s' with no format argument", name);
1181                 return;
1182         }
```

```
1184         do_check_printf_call(caller, name, callexpr, fmtexpr, vaidx);
1185 }


1188 void check_kernel_printf(int id)
1189 {
1190         if (option_project != PROJ_KERNEL)
1191                 return;

1193         my_id = id;

1195 #define printf_hook(func, fmt, first_to_check)  \
1196         add_function_hook(#func, check_printf_call, INT_PTR(fmt + (first_to_chec

1198         /* Extracted using stupid perl script. */

1200 #if 0
1201         printf_hook(srm_printk, 1, 2);                  /* arch/alpha/include/
1202         printf_hook(die_if_kernel, 1, 2);               /* arch/frv/include/as
1203         printf_hook(ia64_mca_printk, 1, 2);             /* arch/ia64/include/a
1204         printf_hook(nfprint, 1, 2);                     /* arch/m68k/include/a
1205         printf_hook(gdbstub_printk, 1, 2);              /* arch/mn10300/includ
1206         printf_hook(DBG, 1, 2);                         /* arch/powerpc/boot/p
1207         printf_hook(printf, 1, 2);                      /* arch/powerpc/boot/s
1208         printf_hook(udbg_printf, 1, 2);                 /* arch/powerpc/includ
1209         printf_hook(__debug_sprintf_event, 3, 4);       /* arch/s390/include/a
1210         printf_hook(__debug_sprintf_exception, 3, 4);   /* arch/s390/include/a
1211         printf_hook(prom_printf, 1, 2);                 /* arch/sparc/include/

1213         printf_hook(fail, 1, 2);                        /* arch/x86/vdso/vdso2
1214 #endif

1216         printf_hook(_ldm_printk, 3, 4);                 /* block/partitions/ld
1217         printf_hook(rbd_warn, 2, 3);                    /* drivers/block/rbd.c
1218         printf_hook(fw_err, 2, 3);                      /* drivers/firewire/co
1219         printf_hook(fw_notice, 2, 3);                   /* drivers/firewire/co
1220         printf_hook(i915_error_printf, 2, 3);           /* drivers/gpu/drm/i91
1221         printf_hook(i915_handle_error, 3, 4);           /* drivers/gpu/drm/i91
1222         printf_hook(nv_printk_, 3, 4);                  /* drivers/gpu/drm/nou
1223         printf_hook(host1x_debug_output, 2, 3);         /* drivers/gpu/host1x/
1224         printf_hook(callc_debug, 2, 3);                 /* drivers/isdn/hisax/
1225         printf_hook(link_debug, 3, 4);                  /* drivers/isdn/hisax/
1226         printf_hook(HiSax_putstatus, 3, 4);             /* drivers/isdn/hisax/
1227         printf_hook(VHiSax_putstatus, 3, 0);            /* drivers/isdn/hisax/
1228         printf_hook(debugl1, 2, 3);                     /* drivers/isdn/hisax/
1229         printf_hook(l3m_debug, 2, 3);                   /* drivers/isdn/hisax/
1230         printf_hook(dout_debug, 2, 3);                  /* drivers/isdn/hisax/
1231         printf_hook(l1m_debug, 2, 3);                   /* drivers/isdn/hisax/
1232         printf_hook(bch_cache_set_error, 2, 3);         /* drivers/md/bcache/b
1233         printf_hook(_tda_printk, 4, 5);                 /* drivers/media/tuner
1234         printf_hook(i40evf_debug_d, 3, 4);              /* drivers/net/etherne
1235         printf_hook(en_print, 3, 4);                    /* drivers/net/etherne
1236         printf_hook(_ath_dbg, 3, 4);                    /* drivers/net/wireles
1237         printf_hook(ath_printk, 3, 4);                  /* drivers/net/wireles
1238         printf_hook(ath10k_dbg, 3, 4);                  /* drivers/net/wireles
1239         printf_hook(ath10k_err, 2, 3);                  /* drivers/net/wireles
1240         printf_hook(ath10k_info, 2, 3);                 /* drivers/net/wireles
1241         printf_hook(ath10k_warn, 2, 3);                 /* drivers/net/wireles
1242         printf_hook(_ath5k_printk, 3, 4);               /* drivers/net/wireles
1243         printf_hook(ATH5K_DBG, 3, 4);                   /* drivers/net/wireles
1244         printf_hook(ATH5K_DBG_UNLIMIT, 3, 4);           /* drivers/net/wireles
1245         printf_hook(ath6kl_printk, 2, 3);               /* drivers/net/wireles
1246         printf_hook(ath6kl_err, 1, 2);                  /* drivers/net/wireles
1247         printf_hook(ath6kl_info, 1, 2);                 /* drivers/net/wireles
1248         printf_hook(ath6kl_warn, 1, 2);                 /* drivers/net/wireles
```



```
1249         printf_hook(wil_dbg_trace, 2, 3);               /* drivers/net/wireles
1250         printf_hook(wil_err, 2, 3);                     /* drivers/net/wireles
1251         printf_hook(wil_err_ratelimited, 2, 3);         /* drivers/net/wireles
1252         printf_hook(wil_info, 2, 3);                    /* drivers/net/wireles
1253         printf_hook(b43dbg, 2, 3);                      /* drivers/net/wireles
1254         printf_hook(b43err, 2, 3);                      /* drivers/net/wireles
1255         printf_hook(b43info, 2, 3);                     /* drivers/net/wireles
1256         printf_hook(b43warn, 2, 3);                     /* drivers/net/wireles
1257         printf_hook(b43legacydbg, 2, 3);                /* drivers/net/wireles
1258         printf_hook(b43legacyerr, 2, 3);                /* drivers/net/wireles
1259         printf_hook(b43legacyinfo, 2, 3);               /* drivers/net/wireles
1260         printf_hook(b43legacywarn, 2, 3);               /* drivers/net/wireles
1261         printf_hook(__brcmf_dbg, 3, 4);                 /* drivers/net/wireles
1262         printf_hook(__brcmf_err, 2, 3);                 /* drivers/net/wireles
1263         printf_hook(__brcms_crit, 2, 3);                /* drivers/net/wireles
1264         printf_hook(__brcms_dbg, 4, 5);                 /* drivers/net/wireles
1265         printf_hook(__brcms_err, 2, 3);                 /* drivers/net/wireles
1266         printf_hook(__brcms_info, 2, 3);                /* drivers/net/wireles
1267         printf_hook(__brcms_warn, 2, 3);                /* drivers/net/wireles
1268         printf_hook(brcmu_dbg_hex_dump, 3, 4);          /* drivers/net/wireles
1269         printf_hook(__iwl_crit, 2, 3);                  /* drivers/net/wireles
1270         printf_hook(__iwl_dbg, 5, 6);                   /* drivers/net/wireles
1271         printf_hook(__iwl_err, 4, 5);                   /* drivers/net/wireles
1272         printf_hook(__iwl_info, 2, 3);                  /* drivers/net/wireles
1273         printf_hook(__iwl_warn, 2, 3);                  /* drivers/net/wireles
1274         printf_hook(rsi_dbg, 2, 3);                     /* drivers/net/wireles
1275         printf_hook(RTPRINT, 4, 5);                     /* drivers/net/wireles
1276         printf_hook(RT_ASSERT, 2, 3);                   /* drivers/net/wireles
1277         printf_hook(RT_TRACE, 4, 5);                    /* drivers/net/wireles
1278         printf_hook(__of_node_dup, 2, 3);               /* drivers/of/of_priva
1279         printf_hook(BNX2FC_HBA_DBG, 2, 3);              /* drivers/scsi/bnx2fc
1280         printf_hook(BNX2FC_IO_DBG, 2, 3);               /* drivers/scsi/bnx2fc
1281         printf_hook(BNX2FC_TGT_DBG, 2, 3);              /* drivers/scsi/bnx2fc
1282         printf_hook(ql_dbg, 4, 5);                      /* drivers/scsi/qla2xx
1283         printf_hook(ql_dbg_pci, 4, 5);                  /* drivers/scsi/qla2xx
1284         printf_hook(ql_log, 4, 5);                      /* drivers/scsi/qla2xx
1285         printf_hook(ql_log_pci, 4, 5);                  /* drivers/scsi/qla2xx
1286         printf_hook(libcfs_debug_msg, 2, 3);            /* drivers/staging/lus
1287         printf_hook(libcfs_debug_vmsg2, 4, 5);          /* drivers/staging/lus
1288         printf_hook(_ldlm_lock_debug, 3, 4);            /* drivers/staging/lus
1289         printf_hook(_debug_req, 3, 4);                  /* drivers/staging/lus
1290         printf_hook(iscsi_change_param_sprintf, 2, 3);  /* drivers/target/iscs
1291         printf_hook(dbg, 1, 2);                         /* drivers/tty/serial/
1292         printf_hook(_usb_stor_dbg, 2, 3);               /* drivers/usb/storage
1293         printf_hook(usb_stor_dbg, 2, 3);                /* drivers/usb/storage
1294         printf_hook(vringh_bad, 1, 2);                  /* drivers/vhost/vring
1295         printf_hook(__adfs_error, 3, 4);                /* fs/adfs/adfs.h */
1296         printf_hook(affs_error, 3, 4);                  /* fs/affs/affs.h */
1297         printf_hook(affs_warning, 3, 4);                /* fs/affs/affs.h */
1298         printf_hook(befs_debug, 2, 3);                  /* fs/befs/befs.h */
1299         printf_hook(befs_error, 2, 3);                  /* fs/befs/befs.h */
1300         printf_hook(befs_warning, 2, 3);                /* fs/befs/befs.h */
1301         printf_hook(__btrfs_panic, 5, 6);               /* fs/btrfs/ctree.h */
1302         printf_hook(__btrfs_std_error, 5, 6);           /* fs/btrfs/ctree.h */
1303         printf_hook(btrfs_printk, 2, 3);                /* fs/btrfs/ctree.h */
1304         printf_hook(cifs_vfs_err, 1, 2);                /* fs/cifs/cifs_debug.
1305         printf_hook(__ecryptfs_printk, 1, 2);           /* fs/ecryptfs/ecryptf
1306         printf_hook(ext2_error, 3, 4);                  /* fs/ext2/ext2.h */
1307         printf_hook(ext2_msg, 3, 4);                    /* fs/ext2/ext2.h */
1308         printf_hook(ext3_abort, 3, 4);                  /* fs/ext3/ext3.h */
1309         printf_hook(ext3_error, 3, 4);                  /* fs/ext3/ext3.h */
1310         printf_hook(ext3_msg, 3, 4);                    /* fs/ext3/ext3.h */
1311         printf_hook(ext3_warning, 3, 4);                /* fs/ext3/ext3.h */
1312         printf_hook(__ext4_abort, 4, 5);                /* fs/ext4/ext4.h */
1313         printf_hook(__ext4_error, 4, 5);                /* fs/ext4/ext4.h */
1314         printf_hook(__ext4_error_file, 5, 6);           /* fs/ext4/ext4.h */
```

```
1315        printf_hook(__ext4_error_inode, 5, 6);          /* fs/ext4/ext4.h */
1316        printf_hook(__ext4_grp_locked_error, 7, 8);     /* fs/ext4/ext4.h */
1317        printf_hook(__ext4_msg, 3, 4);                  /* fs/ext4/ext4.h */
1318        printf_hook(__ext4_warning, 4, 5);              /* fs/ext4/ext4.h */
1319        printf_hook(f2fs_msg, 3, 4);                    /* fs/f2fs/f2fs.h */
1320        printf_hook(__fat_fs_error, 3, 4);              /* fs/fat/fat.h */
1321        printf_hook(fat_msg, 3, 4);                     /* fs/fat/fat.h */
1322        printf_hook(gfs2_print_dbg, 2, 3);              /* fs/gfs2/glock.h */
1323        printf_hook(gfs2_lm_withdraw, 2, 3);            /* fs/gfs2/util.h */
1324        printf_hook(hpfs_error, 2, 3);                  /* fs/hpfs/hpfs_fn.h *
1325        printf_hook(jfs_error, 2, 3);                   /* fs/jfs/jfs_superblo
1326        printf_hook(nilfs_error, 3, 4);                 /* fs/nilfs2/nilfs.h *
1327        printf_hook(nilfs_warning, 3, 4);               /* fs/nilfs2/nilfs.h *
1328        printf_hook(__ntfs_debug, 4, 5);                /* fs/ntfs/debug.h */
1329        printf_hook(__ntfs_error, 3, 4);                /* fs/ntfs/debug.h */
1330        printf_hook(__ntfs_warning, 3, 4);              /* fs/ntfs/debug.h */
1331        printf_hook(__ocfs2_abort, 3, 4);               /* fs/ocfs2/super.h */
1332        printf_hook(__ocfs2_error, 3, 4);               /* fs/ocfs2/super.h */
1333        printf_hook(_udf_err, 3, 4);                    /* fs/udf/udfdecl.h */
1334        printf_hook(_udf_warn, 3, 4);                   /* fs/udf/udfdecl.h */
1335        printf_hook(ufs_error, 3, 4);                   /* fs/ufs/ufs.h */
1336        printf_hook(ufs_panic, 3, 4);                   /* fs/ufs/ufs.h */
1337        printf_hook(ufs_warning, 3, 4);                 /* fs/ufs/ufs.h */
1338        printf_hook(xfs_alert, 2, 3);                   /* fs/xfs/xfs_message.
1339        printf_hook(xfs_alert_tag, 3, 4);               /* fs/xfs/xfs_message.
1340        printf_hook(xfs_crit, 2, 3);                    /* fs/xfs/xfs_message.
1341        printf_hook(xfs_debug, 2, 3);                   /* fs/xfs/xfs_message.
1342        printf_hook(xfs_emerg, 2, 3);                   /* fs/xfs/xfs_message.
1343        printf_hook(xfs_err, 2, 3);                     /* fs/xfs/xfs_message.
1344        printf_hook(xfs_info, 2, 3);                    /* fs/xfs/xfs_message.
1345        printf_hook(xfs_notice, 2, 3);                  /* fs/xfs/xfs_message.
1346        printf_hook(xfs_warn, 2, 3);                    /* fs/xfs/xfs_message.
1347        printf_hook(warn_slowpath_fmt, 3, 4);           /* include/asm-generic
1348        printf_hook(warn_slowpath_fmt_taint, 4, 5);     /* include/asm-generic
1349        printf_hook(drm_err, 1, 2);                     /* include/drm/drmP.h
1350        printf_hook(drm_ut_debug_printk, 2, 3);         /* include/drm/drmP.h
1351        printf_hook(__acpi_handle_debug, 3, 4);         /* include/linux/acpi.
1352        printf_hook(acpi_handle_printk, 3, 4);          /* include/linux/acpi.
1353        printf_hook(audit_log, 4, 5);                   /* include/linux/audit
1354        printf_hook(audit_log_format, 2, 3);            /* include/linux/audit
1355        printf_hook(bdi_register, 3, 4);                /* include/linux/backi
1356        printf_hook(__trace_note_message, 2, 3);        /* include/linux/blktr
1357        printf_hook(_dev_info, 2, 3);                   /* include/linux/devic
1358        printf_hook(dev_alert, 2, 3);                   /* include/linux/devic
1359        printf_hook(dev_crit, 2, 3);                    /* include/linux/devic
1360        printf_hook(dev_emerg, 2, 3);                   /* include/linux/devic
1361        printf_hook(dev_err, 2, 3);                     /* include/linux/devic
1362        printf_hook(dev_notice, 2, 3);                  /* include/linux/devic
1363        printf_hook(dev_printk, 3, 4);                  /* include/linux/devic
1364        printf_hook(dev_printk_emit, 3, 4);             /* include/linux/devic
1365        printf_hook(dev_set_name, 2, 3);                /* include/linux/devic
1366        printf_hook(dev_vprintk_emit, 3, 0);            /* include/linux/devic
1367        printf_hook(dev_warn, 2, 3);                    /* include/linux/devic
1368        printf_hook(device_create, 5, 6);               /* include/linux/devic
1369        printf_hook(device_create_with_groups, 6, 7);   /* include/linux/devic
1370        printf_hook(devm_kasprintf, 3, 4);              /* include/linux/devic
1371        printf_hook(__dynamic_dev_dbg, 3, 4);           /* include/linux/dynam
1372        printf_hook(__dynamic_netdev_dbg, 3, 4);        /* include/linux/dynam
1373        printf_hook(__dynamic_pr_debug, 2, 3);          /* include/linux/dynam
1374        printf_hook(__simple_attr_check_format, 1, 2);  /* include/linux/fs.h
1375        printf_hook(fscache_init_cache, 3, 4);          /* include/linux/fscac
1376        printf_hook(gameport_set_phys, 2, 3);           /* include/linux/gamep
1377        printf_hook(iio_trigger_alloc, 1, 2);           /* include/linux/iio/t
1378        printf_hook(__check_printsym_format, 1, 2);     /* include/linux/kalls
1379        printf_hook(kdb_printf, 1, 2);                  /* include/linux/kdb.h
1380        printf_hook(vkdb_printf, 1, 0);                 /* include/linux/kdb.h
```



```
1381        printf_hook(____trace_printk_check_format, 1, 2);  /* include/linux/kern
1382        printf_hook(__trace_bprintk, 2, 3);             /* include/linux/kerne
1383        printf_hook(__trace_printk, 2, 3);              /* include/linux/kerne
1384        printf_hook(kasprintf, 2, 3);                   /* include/linux/kerne
1385        printf_hook(panic, 1, 2);                       /* include/linux/kerne
1386        printf_hook(scnprintf, 3, 4);                   /* include/linux/kerne
1387        printf_hook(snprintf, 3, 4);                    /* include/linux/kerne
1388        printf_hook(sprintf, 2, 3);                     /* include/linux/kerne
1389        printf_hook(trace_printk, 1, 2);                /* include/linux/kerne
1390        printf_hook(vscnprintf, 3, 0);                  /* include/linux/kerne
1391        printf_hook(vsnprintf, 3, 0);                   /* include/linux/kerne
1392        printf_hook(vsprintf, 2, 0);                    /* include/linux/kerne
1393        printf_hook(vmcoreinfo_append_str, 1, 2);       /* include/linux/kexec
1394        printf_hook(__request_module, 2, 3);            /* include/linux/kmod.
1395        printf_hook(add_uevent_var, 2, 3);              /* include/linux/kobje
1396        printf_hook(kobject_add, 3, 4);                 /* include/linux/kobje
1397        printf_hook(kobject_init_and_add, 4, 5);        /* include/linux/kobje
1398        printf_hook(kobject_set_name, 2, 3);            /* include/linux/kobje
1399        printf_hook(kthread_create_on_node, 4, 5);      /* include/linux/kthre
1400        printf_hook(__ata_ehi_push_desc, 2, 3);         /* include/linux/libat
1401        printf_hook(ata_dev_printk, 3, 4);              /* include/linux/libat
1402        printf_hook(ata_ehi_push_desc, 2, 3);           /* include/linux/libat
1403        printf_hook(ata_link_printk, 3, 4);             /* include/linux/libat
1404        printf_hook(ata_port_desc, 2, 3);               /* include/linux/libat
1405        printf_hook(ata_port_printk, 3, 4);             /* include/linux/libat
1406        printf_hook(warn_alloc_failed, 3, 4);           /* include/linux/mm.h
1407        printf_hook(mmiotrace_printk, 1, 2);            /* include/linux/mmiot
1408        printf_hook(netdev_alert, 2, 3);                /* include/linux/netde
1409        printf_hook(netdev_crit, 2, 3);                 /* include/linux/netde
1410        printf_hook(netdev_emerg, 2, 3);                /* include/linux/netde
1411        printf_hook(netdev_err, 2, 3);                  /* include/linux/netde
1412        printf_hook(netdev_info, 2, 3);                 /* include/linux/netde
1413        printf_hook(netdev_notice, 2, 3);               /* include/linux/netde
1414        printf_hook(netdev_printk, 3, 4);               /* include/linux/netde
1415        printf_hook(netdev_warn, 2, 3);                 /* include/linux/netde
1416        printf_hook(early_printk, 1, 2);                /* include/linux/print
1417        printf_hook(no_printk, 1, 2);                   /* include/linux/print
1418        printf_hook(printk, 1, 2);                      /* include/linux/print
1419        printf_hook(printk_deferred, 1, 2);             /* include/linux/print
1420        printf_hook(printk_emit, 5, 6);                 /* include/linux/print
1421        printf_hook(vprintk, 1, 0);                     /* include/linux/print
1422        printf_hook(vprintk_emit, 5, 0);                /* include/linux/print
1423        printf_hook(__quota_error, 3, 4);               /* include/linux/quota
1424        printf_hook(seq_buf_printf, 2, 3);              /* include/linux/seq_b
1425        printf_hook(seq_buf_vprintf, 2, 0);             /* include/linux/seq_b
1426        printf_hook(seq_printf, 2, 3);                  /* include/linux/seq_f
1427        printf_hook(seq_vprintf, 2, 0);                 /* include/linux/seq_f
1428        printf_hook(bprintf, 3, 4);                     /* include/linux/strin
1429        printf_hook(trace_seq_printf, 2, 3);            /* include/linux/trace
1430        printf_hook(trace_seq_vprintf, 2, 0);           /* include/linux/trace
1431        printf_hook(__alloc_workqueue_key, 1, 6);       /* include/linux/workq
1432        printf_hook(set_worker_desc, 1, 2);             /* include/linux/workq
1433        printf_hook(_p9_debug, 3, 4);                   /* include/net/9p/9p.h
1434        printf_hook(bt_err, 1, 2);                      /* include/net/bluetoo
1435        printf_hook(bt_info, 1, 2);                     /* include/net/bluetoo
1436        printf_hook(nf_ct_helper_log, 3, 4);            /* include/net/netfilt
1437        printf_hook(nf_log_buf_add, 2, 3);              /* include/net/netfilt
1438        printf_hook(nf_log_packet, 8, 9);               /* include/net/netfilt
1439        printf_hook(SOCK_DEBUG, 2, 3);                  /* include/net/sock.h
1440        printf_hook(__snd_printk, 4, 5);                /* include/sound/core.
1441        printf_hook(_snd_printd, 2, 3);                 /* include/sound/core.
1442        printf_hook(snd_printd, 1, 2);                  /* include/sound/core.
1443        printf_hook(snd_printdd, 1, 2);                 /* include/sound/core.
1444        printf_hook(snd_iprintf, 2, 3);                 /* include/sound/info.
1445        printf_hook(snd_seq_create_kernel_client, 3, 4);  /* include/sound/seq_k
1446        printf_hook(xen_raw_printk, 1, 2);              /* include/xen/hvc-con
```

```
1447         printf_hook(xenbus_dev_error, 3, 4);               /* include/xen/xenbus.
1448         printf_hook(xenbus_dev_fatal, 3, 4);               /* include/xen/xenbus.
1449         printf_hook(xenbus_printf, 4, 5);                  /* include/xen/xenbus.
1450         printf_hook(xenbus_watch_pathfmt, 4, 5);           /* include/xen/xenbus.
1451         printf_hook(batadv_fdebug_log, 2, 3);              /* net/batman-adv/debu
1452         printf_hook(_batadv_dbg, 4, 5);                    /* net/batman-adv/main
1453         printf_hook(batadv_debug_log, 2, 3);               /* net/batman-adv/main
1454         printf_hook(__sdata_dbg, 2, 3);                    /* net/mac80211/debug.
1455         printf_hook(__sdata_err, 1, 2);                    /* net/mac80211/debug.
1456         printf_hook(__sdata_info, 1, 2);                   /* net/mac80211/debug.
1457         printf_hook(__wiphy_dbg, 3, 4);                    /* net/mac80211/debug.
1458         printf_hook(mac80211_format_buffer, 4, 5);         /* net/mac80211/debugf
1459         printf_hook(__rds_conn_error, 2, 3);               /* net/rds/rds.h */
1460         printf_hook(rdsdebug, 1, 2);                       /* net/rds/rds.h */
1461         printf_hook(printl, 1, 2);                         /* net/sctp/probe.c */
1462         printf_hook(svc_printk, 2, 3);                     /* net/sunrpc/svc.c */
1463         printf_hook(tomoyo_io_printf, 2, 3);               /* security/tomoyo/com
1464         printf_hook(tomoyo_supervisor, 2, 3);              /* security/tomoyo/com
1465         printf_hook(tomoyo_write_log, 2, 3);               /* security/tomoyo/com
1466         printf_hook(cmp_error, 2, 3);                      /* sound/firewire/cmp.
1467 }
```

```
**********************************************************
    1749 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_kmalloc_to_bugon.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

extern int check_assigned_expr_id;

static int is_kmalloc_call(struct expression *expr)
{
        if (expr->type != EXPR_CALL)
                return 0;
        if (expr->fn->type != EXPR_SYMBOL)
                return 0;
        if (!strcmp(expr->fn->symbol_name->name, "kmalloc"))
                return 1;
        if (!strcmp(expr->fn->symbol_name->name, "kzalloc"))
                return 1;
        return 0;
}

static void match_condition(struct expression *expr)
{
        char *macro;
        struct smatch_state *state;
        struct expression *right;
        char *name;

        macro = get_macro_name(expr->pos);
        if (!macro || strcmp(macro, "BUG_ON") != 0)
                return;
        state = get_state_expr(check_assigned_expr_id, expr);
        if (!state || !state->data)
                return;
        right = (struct expression *)state->data;
        if (!is_kmalloc_call(right))
                return;

        name = expr_to_var(expr);
        sm_warning("bug on allocation failure '%s'", name);
        free_string(name);
}

void check_kmalloc_to_bugon(int id)
{
```



```
        if (option_project != PROJ_KERNEL)
                return;
        if (!option_spammy)
                return;
        my_id = id;
        add_hook(&match_condition, CONDITION_HOOK);
}
```

```
**********************************************************
    2595 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_kmalloc_wrong_size.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static int get_data_size(struct expression *ptr)
{
        struct symbol *type;

        type = get_type(ptr);
        if (!type || type->type != SYM_PTR)
                return 0;
        type = get_base_type(type);
        if (!type)
                return 0;
        return type_bytes(type);
}

static void check_size_matches(int data_size, struct expression *size_expr)
{
        sval_t sval;

        if (data_size == 1)  /* this is generic a buffer */
                return;

        if (!get_implied_value(size_expr, &sval))
                return;
        if (sval_cmp_val(sval, data_size) != 0)
                sm_warning("double check that we're allocating correct size: %d
}

static void match_alloc(const char *fn, struct expression *expr, void *unused)
{
        struct expression *call = strip_expr(expr->right);
        struct expression *arg;
        int ptr_size;

        ptr_size = get_data_size(expr->left);
        if (!ptr_size)
                return;

        arg = get_argument_from_call_expr(call->args, 0);
        arg = strip_expr(arg);
        if (!arg || arg->type != EXPR_BINOP || arg->op != '*')
```



```
                return;
        if (expr->left->type == EXPR_SIZEOF)
                check_size_matches(ptr_size, arg->left);
        if (expr->right->type == EXPR_SIZEOF)
                check_size_matches(ptr_size, arg->right);
}

static void match_calloc(const char *fn, struct expression *expr, void *_arg_nr)
{
        int arg_nr = PTR_INT(_arg_nr);
        struct expression *call = strip_expr(expr->right);
        struct expression *arg;
        int ptr_size;

        ptr_size = get_data_size(expr->left);
        if (!ptr_size)
                return;

        arg = get_argument_from_call_expr(call->args, arg_nr);
        check_size_matches(ptr_size, arg);
}

void check_kmalloc_wrong_size(int id)
{
        my_id = id;

        if (option_project != PROJ_KERNEL) {
                add_function_assign_hook("malloc", &match_alloc, NULL);
                add_function_assign_hook("calloc", &match_calloc, INT_PTR(1));
                return;
        }

        add_function_assign_hook("kmalloc", &match_alloc, NULL);
        add_function_assign_hook("kcalloc", &match_calloc, INT_PTR(1));
}
```

```
**********************************************************
    2342 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_kunmap.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

STATE(no_unmap);

extern int check_assigned_expr_id;
static int my_id;

static void check_assignment(void *data)
{
        struct expression *expr = (struct expression *)data;
        char *fn;

        if (!expr)
                return;
        if (expr->type != EXPR_CALL)
                return;
        fn = expr_to_var(expr->fn);
        if (!fn)
                return;
        if (!strcmp(fn, "kmap"))
                sm_warning("passing the wrong variable to kunmap()");
        free_string(fn);
}

static void match_kmap_atomic(const char *fn, struct expression *expr, void *dat
{
        struct expression *arg;

        arg = get_argument_from_call_expr(expr->args, 0);
        set_state_expr(my_id, arg, &no_unmap);
}

static void match_kunmap_atomic(const char *fn, struct expression *expr, void *d
{
        struct expression *arg;
        struct sm_state *sm;

        arg = get_argument_from_call_expr(expr->args, 0);
        sm = get_sm_state_expr(my_id, arg);
        if (!sm)
                return;
        if (slist_has_state(sm->possible, &no_unmap))
```



```
                sm_warning("passing the wrong variable to kmap_atomic()");
}

static void match_kunmap(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg;
        struct sm_state *sm;
        struct sm_state *tmp;

        arg = get_argument_from_call_expr(expr->args, 0);
        sm = get_sm_state_expr(check_assigned_expr_id, arg);
        if (!sm)
                return;
        FOR_EACH_PTR(sm->possible, tmp) {
                check_assignment(tmp->state->data);
        } END_FOR_EACH_PTR(tmp);
}

void check_kunmap(int id)
{
        my_id = id;
        if (option_project != PROJ_KERNEL)
                return;
        add_function_hook("kunmap", &match_kunmap, NULL);
        add_function_hook("kmap_atomic", &match_kmap_atomic, NULL);
        add_function_hook("kunmap_atomic", &match_kunmap_atomic, NULL);
}
```

```
**********************************************************
    5729 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_leaks.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The point of this check is to look for leaks.
 * foo = malloc();  // <- mark it as allocated.
 * A variable becomes &ok if we:
 * 1) assign it to another variable.
 * 2) pass it to a function.
 *
 * One complication is dealing with stuff like:
 * foo->bar = malloc();
 * foo->baz = malloc();
 * foo = something();
 *
 * The work around is that for now what this check only
 * checks simple expressions and doesn't check whether
 * foo->bar is leaked.
 *
 */

#include <fcntl.h>
#include <unistd.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(allocated);
STATE(ok);

static void set_parent(struct expression *expr, struct smatch_state *state);

static const char *allocation_funcs[] = {
        "malloc",
        "kmalloc",
        "kzalloc",
        "kmemdup",
};

static char *alloc_parent_str(struct symbol *sym)
{
        static char buf[256];

        if (!sym || !sym->ident)
```



```
                return NULL;

        snprintf(buf, 255, "%s", sym->ident->name);
        buf[255] = '\0';
        return alloc_string(buf);
}

static char *get_parent_from_expr(struct expression *expr, struct symbol **sym)
{
        char *name;

        expr = strip_expr(expr);

        name = expr_to_str_sym(expr, sym);
        free_string(name);
        if (!name || !*sym || !(*sym)->ident) {
                *sym = NULL;
                return NULL;
        }
        return alloc_parent_str(*sym);
}

static int is_local(struct expression *expr)
{
        char *name;
        struct symbol *sym;
        int ret = 0;

        name = expr_to_str_sym(expr, &sym);
        if (!name || !sym)
                goto out;
        if (sym->ctype.modifiers & (MOD_NONLOCAL | MOD_STATIC | MOD_ADDRESSABLE))
                goto out;
        ret = 1;
out:
        free_string(name);
        return ret;
}

static int is_param(struct expression *expr)
{
        char *name;
        struct symbol *sym;
        struct symbol *tmp;
        int ret = 0;

        name = expr_to_str_sym(expr, &sym);
        if (!name || !sym)
                goto out;
        FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, tmp) {
                if (tmp == sym) {
                        ret = 1;
                        goto out;
                }
        } END_FOR_EACH_PTR(tmp);
out:
        free_string(name);
        return ret;

}

static void match_alloc(const char *fn, struct expression *expr, void *unused)
{
        if (!is_local(expr->left))
                return;
        if (is_param(expr->left))
```

```
127                 return;
128         if (expr->left->type != EXPR_SYMBOL)
129                 return;
130         set_state_expr(my_id, expr->left, &allocated);
131 }

133 static void match_condition(struct expression *expr)
134 {
135         struct sm_state *sm;

137         expr = strip_expr(expr);

139         switch (expr->type) {
140         case EXPR_PREOP:
141         case EXPR_SYMBOL:
142         case EXPR_DEREF:
143                 sm = get_sm_state_expr(my_id, expr);
144                 if (sm && slist_has_state(sm->possible, &allocated))
145                         set_true_false_states_expr(my_id, expr, NULL, &ok);
146                 return;
147         case EXPR_ASSIGNMENT:
148                  /* You have to deal with stuff like if (a = b = c) */
149                 match_condition(expr->left);
150                 return;
151         default:
152                 return;
153         }
154 }

156 static void set_parent(struct expression *expr, struct smatch_state *state)
157 {
158         char *name;
159         struct symbol *sym;

161         expr = strip_expr(expr);
162         if (!expr)
163                 return;
164         if (expr->type == EXPR_CONDITIONAL ||
165             expr->type == EXPR_SELECT) {
166                 set_parent(expr->cond_true, state);
167                 set_parent(expr->cond_false, state);
168                 return;
169         }

171         name = get_parent_from_expr(expr, &sym);
172         if (!name || !sym)
173                 goto free;
174         if (state == &ok && !get_state(my_id, name, sym))
175                 goto free;
176         set_state(my_id, name, sym, state);
177 free:
178         free_string(name);
179 }

181 static void match_function_call(struct expression *expr)
182 {
183         struct expression *tmp;

185         FOR_EACH_PTR(expr->args, tmp) {
186                 set_parent(tmp, &ok);
187         } END_FOR_EACH_PTR(tmp);
188 }

190 static void warn_if_allocated(struct expression *expr)
191 {
192         struct sm_state *sm;
```



```
193         char *name;
194         sval_t sval;

196         if (get_implied_value(expr, &sval) && sval.value == 0)
197                 return;

199         sm = get_sm_state_expr(my_id, expr);
200         if (!sm)
201                 return;
202         if (!slist_has_state(sm->possible, &allocated))
203                 return;

205         name = expr_to_var(expr);
206         sm_warning("overwrite may leak '%s'", name);
207         free_string(name);

209         /* silence further warnings */
210         set_state_expr(my_id, expr, &ok);
211 }

213 static void match_assign(struct expression *expr)
214 {
215         struct expression *right;

217         right = expr->right;

219         while (right->type == EXPR_ASSIGNMENT)
220                 right = right->left;

222         warn_if_allocated(expr->left);
223         set_parent(right, &ok);
224 }

226 static void check_for_allocated(void)
227 {
228         struct stree *stree;
229         struct sm_state *tmp;

231         stree = __get_cur_stree();
232         FOR_EACH_MY_SM(my_id, stree, tmp) {
233                 if (!slist_has_state(tmp->possible, &allocated))
234                         continue;
235                 sm_warning("possible memory leak of '%s'", tmp->name);
236         } END_FOR_EACH_SM(tmp);
237 }

239 static void match_return(struct expression *ret_value)
240 {
241         if (__inline_fn)
242                 return;
243         set_parent(ret_value, &ok);
244         check_for_allocated();
245 }

247 static void match_end_func(struct symbol *sym)
248 {
249         if (__inline_fn)
250                 return;
251         check_for_allocated();
252 }

254 void check_leaks(int id)
255 {
256         int i;

258         my_id = id;
```

```
260         for (i = 0; i < ARRAY_SIZE(allocation_funcs); i++)
261                 add_function_assign_hook(allocation_funcs[i], &match_alloc, NULL
263         add_hook(&match_condition, CONDITION_HOOK);
265         add_hook(&match_function_call, FUNCTION_CALL_HOOK);
266         add_hook(&match_assign, ASSIGNMENT_HOOK);
268         add_hook(&match_return, RETURN_HOOK);
269         add_hook(&match_end_func, END_FUNC_HOOK);
270 }
```

```
**********************************************************
    5485 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_list.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef CK
#define CK(_x) void _x(int id);
#define __undo_CK_def
#endif

CK(register_db_call_marker) /* always has to be first  */
CK(register_param_used)     /* get_state_hooks have to be registered before smat
CK(register_container_of)
CK(register_container_of2)
CK(register_smatch_extra)   /* smatch_extra always has to be SMATCH_EXTRA */
CK(register_smatch_extra_links)
CK(register_modification_hooks)
/*
 * Implications should probably be after all the modification and smatch_extra
 * hooks have run.
 *
 */
CK(register_implications)
CK(register_definition_db_callbacks)
CK(register_project)
CK(register_untracked_param)
CK(register_buf_comparison)
CK(register_buf_comparison_links)
CK(register_param_compare_limit)
CK(register_param_compare_limit_links)
CK(register_returns_early)

CK(register_smatch_ignore)
CK(register_buf_size)
CK(register_strlen)
CK(register_strlen_equiv)
CK(register_capped)
CK(register_parse_call_math)
CK(register_param_limit)
CK(register_param_filter)
CK(register_param_set)
CK(register_param_cleared)
CK(register_struct_assignment)
CK(register_comparison)
CK(register_comparison_links)
CK(register_comparison_inc_dec)
CK(register_comparison_inc_dec_links)
CK(register_local_values)
CK(register_function_ptrs)
CK(register_annotate)
CK(register_start_states)
CK(register_type_val)
CK(register_data_source)
CK(register_common_functions)
CK(register_function_info)
CK(register_auto_copy)
CK(register_type_links)
CK(register_impossible)
CK(register_impossible_return)
CK(register_strings)
CK(register_real_absolute)
CK(register_imaginary_absolute)
CK(register_fn_arg_link)
CK(register_parameter_names)
CK(register_return_to_param)
```



```
CK(register_return_to_param_links)
CK(register_constraints)
CK(register_constraints_required)
CK(register_about_fn_ptr_arg)
CK(register_mtag)
CK(register_mtag_map)
CK(register_mtag_data)
CK(register_param_to_mtag_data)
CK(register_array_values)
CK(register_nul_terminator)
CK(register_nul_terminator_param_set)
CK(register_statement_count)

CK(register_kernel_user_data2)
CK(register_kernel_user_data3)

CK(check_debug)

CK(check_bogus_loop)

CK(check_deref)
CK(check_check_deref)
CK(check_dereferences_param)
CK(check_index_overflow)
CK(check_index_overflow_loop_marker)
CK(check_testing_index_after_use)
CK(check_memcpy_overflow)
CK(check_strcpy_overflow)
CK(check_sprintf_overflow)
CK(check_snprintf_overflow)
CK(check_allocating_enough_data)
CK(check_leaks)
CK(check_type)
CK(check_allocation_funcs)
CK(check_frees_argument)
CK(check_deref_check)
CK(check_signed)
CK(check_precedence)
CK(check_unused_ret)
CK(check_dma_on_stack)
CK(check_param_mapper)
CK(check_call_tree)
CK(check_dev_queue_xmit)
CK(check_stack)
CK(check_no_return)
CK(check_mod_timer)
CK(check_return)
CK(check_resource_size)
CK(check_release_resource)
CK(check_proc_create)
CK(check_freeing_null)
CK(check_frees_param)
CK(check_free)
CK(check_frees_param_strict)
CK(check_free_strict)
CK(check_no_effect)
CK(check_kunmap)
CK(check_snprintf)
CK(check_macros)
CK(check_return_efault)
CK(check_gfp_dma)
CK(check_unwind)
CK(check_kmalloc_to_bugon)
CK(check_platform_device_put)
CK(check_info_leak)
CK(check_return_enomem)
```

```
CK(check_get_user_overflow)
CK(check_get_user_overflow2)
CK(check_access_ok_math)
CK(check_container_of)
CK(check_input_free_device)
CK(check_select)
CK(check_memset)
CK(check_logical_instead_of_bitwise)
CK(check_kmalloc_wrong_size)
CK(check_pointer_math)
CK(check_bit_shift)
CK(check_macro_side_effects)
CK(check_sizeof)
CK(check_return_cast)
CK(check_or_vs_and)
CK(check_passes_sizeof)
CK(check_assign_vs_compare)
CK(check_missing_break)
CK(check_array_condition)
CK(check_struct_type)
CK(check_64bit_shift)
CK(check_wrong_size_arg)
CK(check_cast_assign)
CK(check_readl_infinite_loops)
CK(check_double_checking)
CK(check_shift_to_zero)
CK(check_indenting)
CK(check_unreachable)
CK(check_no_if_block)
CK(check_buffer_too_small_for_struct)
CK(check_uninitialized)
CK(check_signed_integer_overflow_check)
CK(check_continue_vs_break)
CK(check_impossible_mask)
CK(check_syscall_arg_type)
CK(check_trinity_generator)

/* <- your test goes here */
/* CK(register_template) */

/* kernel specific */
CK(check_kernel_printf)
CK(check_locking)
CK(check_puts_argument)
CK(check_err_ptr)
CK(check_err_ptr_deref)
CK(check_expects_err_ptr)
CK(check_held_dev)
CK(check_return_negative_var)
CK(check_rosenberg)
CK(check_rosenberg2)
CK(check_wait_for_common)
CK(check_bogus_irqrestore)
CK(check_zero_to_err_ptr)
CK(check_freeing_devm)
CK(check_off_by_one_relative)
CK(check_capable)
CK(check_ns_capable)
CK(check_test_bit)
CK(check_dma_mapping_error)
CK(check_nospec)
CK(check_nospec_barrier)
CK(check_spectre)
CK(check_implicit_dependencies)

/* wine specific stuff */
```



```
CK(check_wine_filehandles)
CK(check_wine_WtoA)

/* illumos specific */
CK(check_all_func_returns)

#include "check_list_local.h"

CK(register_scope)
CK(register_stored_conditions)
CK(register_stored_conditions_links)
CK(register_sval)
CK(register_buf_size_late)
CK(register_smatch_extra_late)
CK(register_assigned_expr) /* This is used by smatch_extra.c so it has to come r
CK(register_assigned_expr_links)
CK(register_modification_hooks_late)  /* has to come after smatch_extra */
CK(register_comparison_late) /* has to come after modification_hooks_late */
CK(register_function_hooks)
CK(check_kernel)  /* this is overwriting stuff from smatch_extra_late */
CK(check_wine)
CK(register_returns)

#ifdef __undo_CK_def
#undef CK
#undef __undo_CK_def
#endif
```

```
**********************************************************
   34371 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_locking.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This test checks that locks are held the same across all returns.
 *
 * Of course, some functions are designed to only hold the locks on success.
 * Oh well... We can rewrite it later if we want.
 *
 * The list of wine locking functions came from an earlier script written
 * by Michael Stefaniuc.
 *
 */

#include "parse.h"
#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

static int func_has_transition;

STATE(locked);
STATE(start_state);
STATE(unlocked);
STATE(impossible);

enum action {
        LOCK,
        UNLOCK,
};

enum return_type {
        ret_any,
        ret_non_zero,
        ret_zero,
        ret_negative,
        ret_positive,
};

#define RETURN_VAL -1
#define NO_ARG -2

struct lock_info {
        const char *function;
```



```
        enum action action;
        const char *name;
        int arg;
        enum return_type return_type;
};

static struct lock_info wine_lock_table[] = {
        {"create_window_handle", LOCK, "create_window_handle", RETURN_VAL, ret_n
        {"WIN_GetPtr", LOCK, "create_window_handle", RETURN_VAL, ret_non_zero},
        {"WIN_ReleasePtr", UNLOCK, "create_window_handle", 0, ret_any},
        {"EnterCriticalSection", LOCK, "CriticalSection", 0, ret_any},
        {"LeaveCriticalSection", UNLOCK, "CriticalSection", 0, ret_any},
        {"RtlEnterCriticalSection", LOCK, "RtlCriticalSection", 0, ret_any},
        {"RtlLeaveCriticalSection", UNLOCK, "RtlCriticalSection", 0, ret_any},
        {"GDI_GetObjPtr", LOCK, "GDI_Get", 0, ret_non_zero},
        {"GDI_ReleaseObj", UNLOCK, "GDI_Get", 0, ret_any},
        {"LdrLockLoaderLock", LOCK, "LdrLockLoaderLock", 2, ret_any},
        {"LdrUnlockLoaderLock", UNLOCK, "LdrLockLoaderLock", 1, ret_any},
        {"_lock", LOCK, "_lock", 0, ret_any},
        {"_unlock", UNLOCK, "_lock", 0, ret_any},
        {"msiobj_lock", LOCK, "msiobj_lock", 0, ret_any},
        {"msiobj_unlock", UNLOCK, "msiobj_lock", 0, ret_any},
        {"RtlAcquirePebLock", LOCK, "PebLock", NO_ARG, ret_any},
        {"RtlReleasePebLock", UNLOCK, "PebLock", NO_ARG, ret_any},
        {"server_enter_uninterrupted_section", LOCK, "server_uninterrupted_secti
        {"server_leave_uninterrupted_section", UNLOCK, "server_uninterrupted_sec
        {"RtlLockHeap", LOCK, "RtlLockHeap", 0, ret_any},
        {"RtlUnlockHeap", UNLOCK, "RtlLockHeap", 0, ret_any},
        {"_EnterSysLevel", LOCK, "SysLevel", 0, ret_any},
        {"_LeaveSysLevel", UNLOCK, "SysLevel", 0, ret_any},
        {"USER_Lock", LOCK, "USER_Lock", NO_ARG, ret_any},
        {"USER_Unlock", UNLOCK, "USER_Lock", NO_ARG, ret_any},
        {"wine_tsx11_lock", LOCK, "wine_tsx11_lock", NO_ARG, ret_any},
        {"wine_tsx11_unlock", UNLOCK, "wine_tsx11_lock", NO_ARG, ret_any},
        {"wine_tsx11_lock_ptr", LOCK, "wine_tsx11_lock_ptr", NO_ARG, ret_any},
        {"wine_tsx11_unlock_ptr", UNLOCK, "wine_tsx11_lock_ptr", NO_ARG, ret_any
        {"wined3d_mutex_lock", LOCK, "wined3d_mutex_lock", NO_ARG, ret_any},
        {"wined3d_mutex_unlock", UNLOCK, "wined3d_mutex_lock", NO_ARG, ret_any},
        {"X11DRV_DIB_Lock", LOCK, "X11DRV_DIB_Lock", 0, ret_any},
        {"X11DRV_DIB_Unlock", UNLOCK, "X11DRV_DIB_Lock", 0, ret_any},
};

static struct lock_info kernel_lock_table[] = {
        {"lock_kernel",   LOCK,   "BKL", NO_ARG, ret_any},
        {"unlock_kernel", UNLOCK, "BKL", NO_ARG, ret_any},

        {"spin_lock",                   LOCK,   "spin_lock", 0, ret_any},
        {"spin_unlock",                 UNLOCK, "spin_lock", 0, ret_any},
        {"spin_lock_nested",            LOCK,   "spin_lock", 0, ret_any},
        {"_spin_lock",                  LOCK,   "spin_lock", 0, ret_any},
        {"_spin_unlock",                UNLOCK, "spin_lock", 0, ret_any},
        {"_spin_lock_nested",           LOCK,   "spin_lock", 0, ret_any},
        {"__spin_lock",                 LOCK,   "spin_lock", 0, ret_any},
        {"__spin_unlock",               UNLOCK, "spin_lock", 0, ret_any},
        {"__spin_lock_nested",          LOCK,   "spin_lock", 0, ret_any},
        {"raw_spin_lock",               LOCK,   "spin_lock", 0, ret_any},
        {"raw_spin_unlock",             UNLOCK, "spin_lock", 0, ret_any},
        {"_raw_spin_lock",              LOCK,   "spin_lock", 0, ret_any},
        {"_raw_spin_lock_nested",       LOCK,   "spin_lock", 0, ret_any},
        {"_raw_spin_unlock",            UNLOCK, "spin_lock", 0, ret_any},
        {"__raw_spin_lock",             LOCK,   "spin_lock", 0, ret_any},
        {"__raw_spin_unlock",           UNLOCK, "spin_lock", 0, ret_any},

        {"spin_lock_irq",               LOCK,   "spin_lock", 0, ret_any},
        {"spin_unlock_irq",             UNLOCK, "spin_lock", 0, ret_any},
        {"_spin_lock_irq",              LOCK,   "spin_lock", 0, ret_any},
```

```
127         {"_spin_unlock_irq",           UNLOCK, "spin_lock", 0, ret_any},
128         {"__spin_lock_irq",            LOCK,   "spin_lock", 0, ret_any},
129         {"__spin_unlock_irq",          UNLOCK, "spin_lock", 0, ret_any},
130         {"_raw_spin_lock_irq",         LOCK,   "spin_lock", 0, ret_any},
131         {"_raw_spin_unlock_irq",       UNLOCK, "spin_lock", 0, ret_any},
132         {"__raw_spin_unlock_irq",      UNLOCK, "spin_lock", 0, ret_any},
133         {"spin_lock_irqsave",          LOCK,   "spin_lock", 0, ret_any},
134         {"spin_unlock_irqrestore",     UNLOCK, "spin_lock", 0, ret_any},
135         {"_spin_lock_irqsave",         LOCK,   "spin_lock", 0, ret_any},
136         {"_spin_unlock_irqrestore",    UNLOCK, "spin_lock", 0, ret_any},
137         {"__spin_lock_irqsave",        LOCK,   "spin_lock", 0, ret_any},
138         {"__spin_unlock_irqrestore",   UNLOCK, "spin_lock", 0, ret_any},
139         {"_raw_spin_lock_irqsave",     LOCK,   "spin_lock", 0, ret_any},
140         {"_raw_spin_unlock_irqrestore", UNLOCK, "spin_lock", 0, ret_any},
141         {"__raw_spin_lock_irqsave",    LOCK,   "spin_lock", 0, ret_any},
142         {"__raw_spin_unlock_irqrestore", UNLOCK, "spin_lock", 0, ret_any},
143         {"spin_lock_irqsave_nested",   LOCK,   "spin_lock", 0, ret_any},
144         {"_spin_lock_irqsave_nested",  LOCK,   "spin_lock", 0, ret_any},
145         {"__spin_lock_irqsave_nested", LOCK,   "spin_lock", 0, ret_any},
146         {"_raw_spin_lock_irqsave_nested", LOCK, "spin_lock", 0, ret_any},
147         {"spin_lock_bh",               LOCK,   "spin_lock", 0, ret_any},
148         {"spin_unlock_bh",             UNLOCK, "spin_lock", 0, ret_any},
149         {"_spin_lock_bh",              LOCK,   "spin_lock", 0, ret_any},
150         {"_spin_unlock_bh",            UNLOCK, "spin_lock", 0, ret_any},
151         {"__spin_lock_bh",             LOCK,   "spin_lock", 0, ret_any},
152         {"__spin_unlock_bh",           UNLOCK, "spin_lock", 0, ret_any},

154         {"spin_trylock",               LOCK,   "spin_lock", 0, ret_non_zero},
155         {"_spin_trylock",              LOCK,   "spin_lock", 0, ret_non_zero},
156         {"__spin_trylock",             LOCK,   "spin_lock", 0, ret_non_zero},
157         {"raw_spin_trylock",           LOCK,   "spin_lock", 0, ret_non_zero},
158         {"_raw_spin_trylock",          LOCK,   "spin_lock", 0, ret_non_zero},
159         {"spin_trylock_irq",           LOCK,   "spin_lock", 0, ret_non_zero},
160         {"spin_trylock_irqsave",       LOCK,   "spin_lock", 0, ret_non_zero},
161         {"spin_trylock_bh",            LOCK,   "spin_lock", 0, ret_non_zero},
162         {"_spin_trylock_bh",           LOCK,   "spin_lock", 0, ret_non_zero},
163         {"__spin_trylock_bh",          LOCK,   "spin_lock", 0, ret_non_zero},
164         {"__raw_spin_trylock",         LOCK,   "spin_lock", 0, ret_non_zero},
165         {"_atomic_dec_and_lock",       LOCK,   "spin_lock", 1, ret_non_zero},

167         {"read_lock",                 LOCK,   "read_lock", 0, ret_any},
168         {"read_unlock",               UNLOCK, "read_lock", 0, ret_any},
169         {"_read_lock",                LOCK,   "read_lock", 0, ret_any},
170         {"_read_unlock",              UNLOCK, "read_lock", 0, ret_any},
171         {"__read_lock",               LOCK,   "read_lock", 0, ret_any},
172         {"__read_unlock",             UNLOCK, "read_lock", 0, ret_any},
173         {"_raw_read_lock",            LOCK,   "read_lock", 0, ret_any},
174         {"_raw_read_unlock",          UNLOCK, "read_lock", 0, ret_any},
175         {"__raw_read_lock",           LOCK,   "read_lock", 0, ret_any},
176         {"__raw_read_unlock",         UNLOCK, "read_lock", 0, ret_any},
177         {"read_lock_irq",             LOCK,   "read_lock", 0, ret_any},
178         {"read_unlock_irq" ,          UNLOCK, "read_lock", 0, ret_any},
179         {"_read_lock_irq",            LOCK,   "read_lock", 0, ret_any},
180         {"_read_unlock_irq",          UNLOCK, "read_lock", 0, ret_any},
181         {"__read_lock_irq",           LOCK,   "read_lock", 0, ret_any},
182         {"__read_unlock_irq",         UNLOCK, "read_lock", 0, ret_any},
183         {"read_lock_irqsave",         LOCK,   "read_lock", 0, ret_any},
184         {"read_unlock_irqrestore",    UNLOCK, "read_lock", 0, ret_any},
185         {"_read_lock_irqsave",        LOCK,   "read_lock", 0, ret_any},
186         {"_read_unlock_irqrestore",   UNLOCK, "read_lock", 0, ret_any},
187         {"__read_lock_irqsave",       LOCK,   "read_lock", 0, ret_any},
188         {"__read_unlock_irqrestore",  UNLOCK, "read_lock", 0, ret_any},
189         {"read_lock_bh",              LOCK,   "read_lock", 0, ret_any},
190         {"read_unlock_bh",            UNLOCK, "read_lock", 0, ret_any},
191         {"_read_lock_bh",             LOCK,   "read_lock", 0, ret_any},
192         {"_read_unlock_bh",           UNLOCK, "read_lock", 0, ret_any},
```



```
193         {"__read_lock_bh",            LOCK,   "read_lock", 0, ret_any},
194         {"__read_unlock_bh",          UNLOCK, "read_lock", 0, ret_any},
195         {"_raw_read_lock_bh",         LOCK,   "read_lock", 0, ret_any},
196         {"_raw_read_unlock_bh",       UNLOCK, "read_lock", 0, ret_any},
197         {"__raw_read_lock_bh",        LOCK,   "read_lock", 0, ret_any},
198         {"__raw_read_unlock_bh",      UNLOCK, "read_lock", 0, ret_any},

200         {"generic__raw_read_trylock", LOCK,   "read_lock", 0, ret_non_zero},
201         {"read_trylock",              LOCK,   "read_lock", 0, ret_non_zero},
202         {"_read_trylock",             LOCK,   "read_lock", 0, ret_non_zero},
203         {"raw_read_trylock",          LOCK,   "read_lock", 0, ret_non_zero},
204         {"_raw_read_trylock",         LOCK,   "read_lock", 0, ret_non_zero},
205         {"__raw_read_trylock",        LOCK,   "read_lock", 0, ret_non_zero},
206         {"__read_trylock",            LOCK,   "read_lock", 0, ret_non_zero},

208         {"write_lock",                LOCK,   "write_lock", 0, ret_any},
209         {"write_unlock",              UNLOCK, "write_lock", 0, ret_any},
210         {"_write_lock",               LOCK,   "write_lock", 0, ret_any},
211         {"_write_unlock",             UNLOCK, "write_lock", 0, ret_any},
212         {"__write_lock",              LOCK,   "write_lock", 0, ret_any},
213         {"__write_unlock",            UNLOCK, "write_lock", 0, ret_any},
214         {"write_lock_irq",            LOCK,   "write_lock", 0, ret_any},
215         {"write_unlock_irq",          UNLOCK, "write_lock", 0, ret_any},
216         {"_write_lock_irq",           LOCK,   "write_lock", 0, ret_any},
217         {"_write_unlock_irq",         UNLOCK, "write_lock", 0, ret_any},
218         {"__write_lock_irq",          LOCK,   "write_lock", 0, ret_any},
219         {"__write_unlock_irq",        UNLOCK, "write_lock", 0, ret_any},
220         {"write_lock_irqsave",        LOCK,   "write_lock", 0, ret_any},
221         {"write_unlock_irqrestore",   UNLOCK, "write_lock", 0, ret_any},
222         {"_write_lock_irqsave",       LOCK,   "write_lock", 0, ret_any},
223         {"_write_unlock_irqrestore",  UNLOCK, "write_lock", 0, ret_any},
224         {"__write_lock_irqsave",      LOCK,   "write_lock", 0, ret_any},
225         {"__write_unlock_irqrestore", UNLOCK, "write_lock", 0, ret_any},
226         {"write_lock_bh",             LOCK,   "write_lock", 0, ret_any},
227         {"write_unlock_bh",           UNLOCK, "write_lock", 0, ret_any},
228         {"_write_lock_bh",            LOCK,   "write_lock", 0, ret_any},
229         {"_write_unlock_bh",          UNLOCK, "write_lock", 0, ret_any},
230         {"__write_lock_bh",           LOCK,   "write_lock", 0, ret_any},
231         {"__write_unlock_bh",         UNLOCK, "write_lock", 0, ret_any},
232         {"_raw_write_lock",           LOCK,   "write_lock", 0, ret_any},
233         {"__raw_write_lock",          LOCK,   "write_lock", 0, ret_any},
234         {"_raw_write_unlock",         UNLOCK, "write_lock", 0, ret_any},
235         {"__raw_write_unlock",        UNLOCK, "write_lock", 0, ret_any},

237         {"write_trylock",             LOCK,   "write_lock", 0, ret_non_zero},
238         {"_write_trylock",            LOCK,   "write_lock", 0, ret_non_zero},
239         {"raw_write_trylock",         LOCK,   "write_lock", 0, ret_non_zero},
240         {"_raw_write_trylock",        LOCK,   "write_lock", 0, ret_non_zero},
241         {"__write_trylock",           LOCK,   "write_lock", 0, ret_non_zero},
242         {"__raw_write_trylock",       LOCK,   "write_lock", 0, ret_non_zero},

244         {"down",               LOCK,   "sem", 0, ret_any},
245         {"up",                 UNLOCK, "sem", 0, ret_any},
246         {"down_trylock",       LOCK,   "sem", 0, ret_zero},
247         {"down_timeout",       LOCK,   "sem", 0, ret_zero},
248         {"down_interruptible", LOCK,   "sem", 0, ret_zero},

250         {"mutex_lock",                     LOCK,   "mutex", 0, ret_any},
251         {"mutex_unlock",                   UNLOCK, "mutex", 0, ret_any},
252         {"mutex_lock_nested",              LOCK,   "mutex", 0, ret_any},

254         {"mutex_lock_interruptible",       LOCK,   "mutex", 0, ret_zero},
255         {"mutex_lock_interruptible_nested", LOCK,   "mutex", 0, ret_zero},
256         {"mutex_lock_killable",            LOCK,   "mutex", 0, ret_zero},
257         {"mutex_lock_killable_nested",     LOCK,   "mutex", 0, ret_zero},
```

```
259         {"mutex_trylock",                   LOCK,   "mutex", 0, ret_non_zero},

261         {"raw_local_irq_disable", LOCK,   "irq", NO_ARG, ret_any},
262         {"raw_local_irq_enable",  UNLOCK, "irq", NO_ARG, ret_any},
263         {"spin_lock_irq",         LOCK,   "irq", NO_ARG, ret_any},
264         {"spin_unlock_irq",       UNLOCK, "irq", NO_ARG, ret_any},
265         {"_spin_lock_irq",        LOCK,   "irq", NO_ARG, ret_any},
266         {"_spin_unlock_irq",      UNLOCK, "irq", NO_ARG, ret_any},
267         {"__spin_lock_irq",       LOCK,   "irq", NO_ARG, ret_any},
268         {"__spin_unlock_irq",     UNLOCK, "irq", NO_ARG, ret_any},
269         {"_raw_spin_lock_irq",    LOCK,   "irq", NO_ARG, ret_any},
270         {"_raw_spin_unlock_irq",  UNLOCK, "irq", NO_ARG, ret_any},
271         {"__raw_spin_unlock_irq", UNLOCK, "irq", NO_ARG, ret_any},
272         {"spin_trylock_irq",      LOCK,   "irq", NO_ARG, ret_non_zero},
273         {"read_lock_irq",         LOCK,   "irq", NO_ARG, ret_any},
274         {"read_unlock_irq",       UNLOCK, "irq", NO_ARG, ret_any},
275         {"_read_lock_irq",        LOCK,   "irq", NO_ARG, ret_any},
276         {"_read_unlock_irq",      UNLOCK, "irq", NO_ARG, ret_any},
277         {"__read_lock_irq",       LOCK,   "irq", NO_ARG, ret_any},
278         {"__read_unlock_irq",     UNLOCK, "irq", NO_ARG, ret_any},
279         {"write_lock_irq",        LOCK,   "irq", NO_ARG, ret_any},
280         {"write_unlock_irq",      UNLOCK, "irq", NO_ARG, ret_any},
281         {"_write_lock_irq",       LOCK,   "irq", NO_ARG, ret_any},
282         {"_write_unlock_irq",     UNLOCK, "irq", NO_ARG, ret_any},
283         {"__write_lock_irq",      LOCK,   "irq", NO_ARG, ret_any},
284         {"__write_unlock_irq",    UNLOCK, "irq", NO_ARG, ret_any},

286         {"arch_local_irq_save",         LOCK,   "irqsave", RETURN_VAL, ret_any},
287         {"arch_local_irq_restore",      UNLOCK, "irqsave", 0, ret_any},
288         {"__raw_local_irq_save",        LOCK,   "irqsave", RETURN_VAL, ret_any},
289         {"raw_local_irq_restore",       UNLOCK, "irqsave", 0, ret_any},
290         {"spin_lock_irqsave_nested",    LOCK,   "irqsave", RETURN_VAL, ret_any},
291         {"spin_lock_irqsave",           LOCK,   "irqsave", RETURN_VAL, ret_any},
292         {"spin_lock_irqsave",           LOCK,   "irqsave", 1, ret_any},
293         {"spin_unlock_irqrestore",      UNLOCK, "irqsave", 1, ret_any},
294         {"_spin_lock_irqsave_nested",   LOCK,   "irqsave", RETURN_VAL, ret_any},
295         {"_spin_lock_irqsave",          LOCK,   "irqsave", RETURN_VAL, ret_any},
296         {"_spin_lock_irqsave",          LOCK,   "irqsave", 1, ret_any},
297         {"_spin_unlock_irqrestore",     UNLOCK, "irqsave", 1, ret_any},
298         {"__spin_lock_irqsave_nested",  LOCK,   "irqsave", 1, ret_any},
299         {"__spin_lock_irqsave",         LOCK,   "irqsave", 1, ret_any},
300         {"__spin_unlock_irqrestore",    UNLOCK, "irqsave", 1, ret_any},
301         {"_raw_spin_lock_irqsave",      LOCK,   "irqsave", RETURN_VAL, ret_any},
302         {"_raw_spin_lock_irqsave",      LOCK,   "irqsave", 1, ret_any},
303         {"_raw_spin_unlock_irqrestore",UNLOCK, "irqsave", 1, ret_any},
304         {"__raw_spin_lock_irqsave",     LOCK,   "irqsave", RETURN_VAL, ret_any},
305         {"__raw_spin_unlock_irqrestore",UNLOCK, "irqsave", 1, ret_any},
306         {"_raw_spin_lock_irqsave_nested", LOCK, "irqsave", RETURN_VAL, ret_any},
307         {"spin_trylock_irqsave",        LOCK,   "irqsave", 1, ret_non_zero},
308         {"read_lock_irqsave",           LOCK,   "irqsave", RETURN_VAL, ret_any},
309         {"read_lock_irqsave",           LOCK,   "irqsave", 1, ret_any},
310         {"read_unlock_irqrestore",      UNLOCK, "irqsave", 1, ret_any},
311         {"_read_lock_irqsave",          LOCK,   "irqsave", RETURN_VAL, ret_any},
312         {"_read_lock_irqsave",          LOCK,   "irqsave", 1, ret_any},
313         {"_read_unlock_irqrestore",     UNLOCK, "irqsave", 1, ret_any},
314         {"__read_lock_irqsave",         LOCK,   "irqsave", RETURN_VAL, ret_any},
315         {"__read_unlock_irqrestore",    UNLOCK, "irqsave", 1, ret_any},
316         {"write_lock_irqsave",          LOCK,   "irqsave", RETURN_VAL, ret_any},
317         {"write_lock_irqsave",          LOCK,   "irqsave", 1, ret_any},
318         {"write_unlock_irqrestore",     UNLOCK, "irqsave", 1, ret_any},
319         {"_write_lock_irqsave",         LOCK,   "irqsave", RETURN_VAL, ret_any},
320         {"_write_lock_irqsave",         LOCK,   "irqsave", 1, ret_any},
321         {"_write_unlock_irqrestore",    UNLOCK, "irqsave", 1, ret_any},
322         {"__write_lock_irqsave",        LOCK,   "irqsave", RETURN_VAL, ret_any},
323         {"__write_unlock_irqrestore",   UNLOCK, "irqsave", 1, ret_any},
```



```
325         {"local_bh_disable",    LOCK,   "bottom_half", NO_ARG, ret_any},
326         {"_local_bh_disable",   LOCK,   "bottom_half", NO_ARG, ret_any},
327         {"__local_bh_disable",  LOCK,   "bottom_half", NO_ARG, ret_any},
328         {"local_bh_enable",     UNLOCK, "bottom_half", NO_ARG, ret_any},
329         {"_local_bh_enable",    UNLOCK, "bottom_half", NO_ARG, ret_any},
330         {"__local_bh_enable",   UNLOCK, "bottom_half", NO_ARG, ret_any},
331         {"spin_lock_bh",        LOCK,   "bottom_half", NO_ARG, ret_any},
332         {"spin_unlock_bh",      UNLOCK, "bottom_half", NO_ARG, ret_any},
333         {"_spin_lock_bh",       LOCK,   "bottom_half", NO_ARG, ret_any},
334         {"_spin_unlock_bh",     UNLOCK, "bottom_half", NO_ARG, ret_any},
335         {"__spin_lock_bh",      LOCK,   "bottom_half", NO_ARG, ret_any},
336         {"__spin_unlock_bh",    UNLOCK, "bottom_half", NO_ARG, ret_any},
337         {"read_lock_bh",        LOCK,   "bottom_half", NO_ARG, ret_any},
338         {"read_unlock_bh",      UNLOCK, "bottom_half", NO_ARG, ret_any},
339         {"_read_lock_bh",       LOCK,   "bottom_half", NO_ARG, ret_any},
340         {"_read_unlock_bh",     UNLOCK, "bottom_half", NO_ARG, ret_any},
341         {"__read_lock_bh",      LOCK,   "bottom_half", NO_ARG, ret_any},
342         {"__read_unlock_bh",    UNLOCK, "bottom_half", NO_ARG, ret_any},
343         {"_raw_read_lock_bh",   LOCK,   "bottom_half", NO_ARG, ret_any},
344         {"_raw_read_unlock_bh", UNLOCK, "bottom_half", NO_ARG, ret_any},
345         {"write_lock_bh",       LOCK,   "bottom_half", NO_ARG, ret_any},
346         {"write_unlock_bh",     UNLOCK, "bottom_half", NO_ARG, ret_any},
347         {"_write_lock_bh",      LOCK,   "bottom_half", NO_ARG, ret_any},
348         {"_write_unlock_bh",    UNLOCK, "bottom_half", NO_ARG, ret_any},
349         {"__write_lock_bh",     LOCK,   "bottom_half", NO_ARG, ret_any},
350         {"__write_unlock_bh",   UNLOCK, "bottom_half", NO_ARG, ret_any},
351         {"spin_trylock_bh",     LOCK,   "bottom_half", NO_ARG, ret_non_zero},
352         {"_spin_trylock_bh",    LOCK,   "bottom_half", NO_ARG, ret_non_zero},
353         {"__spin_trylock_bh",   LOCK,   "bottom_half", NO_ARG, ret_non_zero},

355         {"ffs_mutex_lock",        LOCK,   "mutex", 0, ret_zero},
356 };

358 static struct lock_info *lock_table;

360 static struct tracker_list *starts_locked;
361 static struct tracker_list *starts_unlocked;

363 struct locks_on_return {
364         int line;
365         struct tracker_list *locked;
366         struct tracker_list *unlocked;
367         struct tracker_list *impossible;
368         struct range_list *return_values;
369 };
370 DECLARE_PTR_LIST(return_list, struct locks_on_return);
371 static struct return_list *all_returns;

373 static char *make_full_name(const char *lock, const char *var)
374 {
375         static char tmp_buf[512];

377         snprintf(tmp_buf, sizeof(tmp_buf), "%s:%s", lock, var);
378         remove_parens(tmp_buf);
379         return alloc_string(tmp_buf);
380 }

382 static struct expression *remove_spinlock_check(struct expression *expr)
383 {
384         if (expr->type != EXPR_CALL)
385                 return expr;
386         if (expr->fn->type != EXPR_SYMBOL)
387                 return expr;
388         if (strcmp(expr->fn->symbol_name->name, "spinlock_check"))
389                 return expr;
390         expr = get_argument_from_call_expr(expr->args, 0);
```

```
391         return expr;
392 }

394 static char *get_full_name(struct expression *expr, int index)
395 {
396         struct expression *arg;
397         char *name = NULL;
398         char *full_name = NULL;
399         struct lock_info *lock = &lock_table[index];

401         if (lock->arg == RETURN_VAL) {
402                 name = expr_to_var(expr->left);
403                 full_name = make_full_name(lock->name, name);
404         } else if (lock->arg == NO_ARG) {
405                 full_name = make_full_name(lock->name, "");
406         } else {
407                 arg = get_argument_from_call_expr(expr->args, lock->arg);
408                 if (!arg)
409                         goto free;
410                 arg = remove_spinlock_check(arg);
411                 name = expr_to_str(arg);
412                 if (!name)
413                         goto free;
414                 full_name = make_full_name(lock->name, name);
415         }
416 free:
417         free_string(name);
418         return full_name;
419 }

421 static struct smatch_state *get_start_state(struct sm_state *sm)
422 {
423         int is_locked = 0;
424         int is_unlocked = 0;

426         if (in_tracker_list(starts_locked, my_id, sm->name, sm->sym))
427                 is_locked = 1;
428         if (in_tracker_list(starts_unlocked, my_id, sm->name, sm->sym))
429                 is_unlocked = 1;
430         if (is_locked && is_unlocked)
431                 return &undefined;
432         if (is_locked)
433                 return &locked;
434         if (is_unlocked)
435                 return &unlocked;
436         return &undefined;
437 }

439 static struct smatch_state *unmatched_state(struct sm_state *sm)
440 {
441         return &start_state;
442 }

444 static void pre_merge_hook(struct sm_state *sm)
445 {
446         if (is_impossible_path())
447                 set_state(my_id, sm->name, sm->sym, &impossible);
448 }

450 static void do_lock(const char *name)
451 {
452         struct sm_state *sm;

454         if (__inline_fn)
455                 return;
```



```
457         sm = get_sm_state(my_id, name, NULL);
458         if (!sm)
459                 add_tracker(&starts_unlocked, my_id, name, NULL);
460         if (sm && slist_has_state(sm->possible, &locked) &&
461                         strcmp(name, "bottom_half:") != 0)
462                 sm_error("double lock '%s'", name);
463         if (sm)
464                 func_has_transition = TRUE;
465         set_state(my_id, name, NULL, &locked);
466 }

468 static void do_lock_failed(const char *name)
469 {
470         struct sm_state *sm;

472         if (__inline_fn)
473                 return;

475         sm = get_sm_state(my_id, name, NULL);
476         if (!sm)
477                 add_tracker(&starts_unlocked, my_id, name, NULL);
478         set_state(my_id, name, NULL, &unlocked);
479 }

481 static void do_unlock(const char *name)
482 {
483         struct sm_state *sm;

485         if (__inline_fn)
486                 return;
487         if (__path_is_null())
488                 return;
489         sm = get_sm_state(my_id, name, NULL);
490         if (!sm)
491                 add_tracker(&starts_locked, my_id, name, NULL);
492         if (sm && slist_has_state(sm->possible, &unlocked) &&
493                         strcmp(name, "bottom_half:") != 0)
494                 sm_error("double unlock '%s'", name);
495         if (sm)
496                 func_has_transition = TRUE;
497         set_state(my_id, name, NULL, &unlocked);
498 }

500 static void match_lock_held(const char *fn, struct expression *call_expr,
501                         struct expression *assign_expr, void *_index)
502 {
503         int index = PTR_INT(_index);
504         char *lock_name;
505         struct lock_info *lock = &lock_table[index];

507         if (lock->arg == NO_ARG) {
508                 lock_name = get_full_name(NULL, index);
509         } else if (lock->arg == RETURN_VAL) {
510                 if (!assign_expr)
511                         return;
512                 lock_name = get_full_name(assign_expr, index);
513         } else {
514                 lock_name = get_full_name(call_expr, index);
515         }
516         if (!lock_name)
517                 return;
518         do_lock(lock_name);
519         free_string(lock_name);
520 }

522 static void match_lock_failed(const char *fn, struct expression *call_expr,
```

```
523                         struct expression *assign_expr, void *_index)
524 {
525         int index = PTR_INT(_index);
526         char *lock_name;
527         struct lock_info *lock = &lock_table[index];
528
529         if (lock->arg == NO_ARG) {
530                 lock_name = get_full_name(NULL, index);
531         } else if (lock->arg == RETURN_VAL) {
532                 if (!assign_expr)
533                         return;
534                 lock_name = get_full_name(assign_expr, index);
535         } else {
536                 lock_name = get_full_name(call_expr, index);
537         }
538         if (!lock_name)
539                 return;
540         do_lock_failed(lock_name);
541         free_string(lock_name);
542 }
543
544 static void match_returns_locked(const char *fn, struct expression *expr,
545                                  void *_index)
546 {
547         char *full_name = NULL;
548         int index = PTR_INT(_index);
549         struct lock_info *lock = &lock_table[index];
550
551         if (lock->arg != RETURN_VAL)
552                 return;
553         full_name = get_full_name(expr, index);
554         do_lock(full_name);
555 }
556
557 static void match_lock_unlock(const char *fn, struct expression *expr, void *_in
558 {
559         char *full_name = NULL;
560         int index = PTR_INT(_index);
561         struct lock_info *lock = &lock_table[index];
562
563         if (__inline_fn)
564                 return;
565
566         full_name = get_full_name(expr, index);
567         if (!full_name)
568                 return;
569         if (lock->action == LOCK)
570                 do_lock(full_name);
571         else
572                 do_unlock(full_name);
573         free_string(full_name);
574 }
575
576 static struct locks_on_return *alloc_return(struct expression *expr)
577 {
578         struct locks_on_return *ret;
579
580         ret = malloc(sizeof(*ret));
581         if (!get_implied_rl(expr, &ret->return_values))
582                 ret->return_values = NULL;
583         ret->line = get_lineno();
584         ret->locked = NULL;
585         ret->unlocked = NULL;
586         ret->impossible = NULL;
587         return ret;
588 }
```



```
590 static int check_possible(struct sm_state *sm)
591 {
592         struct sm_state *tmp;
593         int islocked = 0;
594         int isunlocked = 0;
595         int undef = 0;
596
597         if (!option_spammy)
598                 return 0;
599
600         FOR_EACH_PTR(sm->possible, tmp) {
601                 if (tmp->state == &locked)
602                         islocked = 1;
603                 if (tmp->state == &unlocked)
604                         isunlocked = 1;
605                 if (tmp->state == &start_state) {
606                         struct smatch_state *s;
607
608                         s = get_start_state(tmp);
609                         if (s == &locked)
610                                 islocked = 1;
611                         else if (s == &unlocked)
612                                 isunlocked = 1;
613                         else
614                                 undef = 1;
615                 }
616                 if (tmp->state == &undefined)
617                         undef = 1;  // i don't think this is possible any more.
618         } END_FOR_EACH_PTR(tmp);
619         if ((islocked && isunlocked) || undef) {
620                 sm_warning("'%s' is sometimes locked here and sometimes unlocked
621                 return 1;
622         }
623         return 0;
624 }
625
626 static struct position warned_pos;
627
628 static void match_return(int return_id, char *return_ranges, struct expression *
629 {
630         struct locks_on_return *ret;
631         struct stree *stree;
632         struct sm_state *tmp;
633
634         if (!final_pass)
635                 return;
636         if (__inline_fn)
637                 return;
638
639         if (expr && cmp_pos(expr->pos, warned_pos) == 0)
640                 return;
641
642         ret = alloc_return(expr);
643
644         stree = __get_cur_stree();
645         FOR_EACH_MY_SM(my_id, stree, tmp) {
646                 if (tmp->state == &locked) {
647                         add_tracker(&ret->locked, tmp->owner, tmp->name,
648                                 tmp->sym);
649                 } else if (tmp->state == &unlocked) {
650                         add_tracker(&ret->unlocked, tmp->owner, tmp->name,
651                                 tmp->sym);
652                 } else if (tmp->state == &start_state) {
653                         struct smatch_state *s;
```

```
655                         s = get_start_state(tmp);
656                         if (s == &locked)
657                                 add_tracker(&ret->locked, tmp->owner, tmp->name,
658                                             tmp->sym);
659                         if (s == &unlocked)
660                                 add_tracker(&ret->unlocked, tmp->owner,tmp->name
661                                             tmp->sym);
662                 } else if (tmp->state == &impossible) {
663                         add_tracker(&ret->impossible, tmp->owner, tmp->name,
664                                     tmp->sym);
665                 } else {
666                         if (check_possible(tmp)) {
667                                 if (expr)
668                                         warned_pos = expr->pos;
669                         }
670                 }
671         } END_FOR_EACH_SM(tmp);
672         add_ptr_list(&all_returns, ret);
673 }

675 static void add_line(struct range_list **rl, int line)
676 {
677         sval_t sval = sval_type_val(&int_ctype, line);

679         add_range(rl, sval, sval);
680 }

682 static int line_printed(struct range_list *rl, int line)
683 {
684         sval_t sval = sval_type_val(&int_ctype, line);

686         return rl_has_sval(rl, sval);
687 }

689 static void print_inconsistent_returns(struct tracker *lock,
690                                        struct smatch_state *start)
691 {
692         struct locks_on_return *tmp;
693         struct range_list *printed = NULL;
694         int i;

696         sm_warning("inconsistent returns '%s'.", lock->name);
697         sm_printf("  Locked on:   ");

699         i = 0;
700         FOR_EACH_PTR(all_returns, tmp) {
701                 if (line_printed(printed, tmp->line))
702                         continue;
703                 if (in_tracker_list(tmp->unlocked, lock->owner, lock->name, lock
704                         continue;
705                 if (in_tracker_list(tmp->locked, lock->owner, lock->name, lock->
706                         if (i++)
707                                 sm_printf("               ");
708                         sm_printf("line %d\n", tmp->line);
709                         add_line(&printed, tmp->line);
710                         continue;
711                 }
712                 if (start == &locked) {
713                         if (i++)
714                                 sm_printf("               ");
715                         sm_printf("line %d\n", tmp->line);
716                         add_line(&printed, tmp->line);
717                 }
718         } END_FOR_EACH_PTR(tmp);

720         sm_printf("  Unlocked on: ");
```



```
721         printed = NULL;
722         i = 0;
723         FOR_EACH_PTR(all_returns, tmp) {
724                 if (line_printed(printed, tmp->line))
725                         continue;
726                 if (in_tracker_list(tmp->unlocked, lock->owner, lock->name, lock
727                         if (i++)
728                                 sm_printf("               ");
729                         sm_printf("line %d\n", tmp->line);
730                         add_line(&printed, tmp->line);
731                         continue;
732                 }
733                 if (in_tracker_list(tmp->locked, lock->owner, lock->name, lock->
734                         continue;
735                 if (start == &unlocked) {
736                         if (i++)
737                                 sm_printf("               ");
738                         sm_printf("line %d\n", tmp->line);
739                         add_line(&printed, tmp->line);
740                 }
741         } END_FOR_EACH_PTR(tmp);
742 }

744 static int matches_return_type(struct range_list *rl, enum return_type type)
745 {
746         sval_t zero_sval = ll_to_sval(0);

748         /* All these double negatives are super ugly!  */

750         switch (type) {
751         case ret_zero:
752                 return !possibly_true_rl(rl, SPECIAL_NOTEQUAL, alloc_rl(zero_sva
753         case ret_non_zero:
754                 return !possibly_true_rl(rl, SPECIAL_EQUAL, alloc_rl(zero_sval,
755         case ret_negative:
756                 return !possibly_true_rl(rl, SPECIAL_GTE, alloc_rl(zero_sval, ze
757         case ret_positive:
758                 return !possibly_true_rl(rl, '<', alloc_rl(zero_sval, zero_sval)
759         case ret_any:
760         default:
761                 return 1;
762         }
763 }

765 static int match_held(struct tracker *lock, struct locks_on_return *this_return,
766 {
767         if (in_tracker_list(this_return->impossible, lock->owner, lock->name, lo
768                 return 0;
769         if (in_tracker_list(this_return->unlocked, lock->owner, lock->name, lock
770                 return 0;
771         if (in_tracker_list(this_return->locked, lock->owner, lock->name, lock->
772                 return 1;
773         if (start == &unlocked)
774                 return 0;
775         return 1;
776 }

778 static int match_released(struct tracker *lock, struct locks_on_return *this_ret
779 {
780         if (in_tracker_list(this_return->impossible, lock->owner, lock->name, lo
781                 return 0;
782         if (in_tracker_list(this_return->unlocked, lock->owner, lock->name, lock
783                 return 1;
784         if (in_tracker_list(this_return->locked, lock->owner, lock->name, lock->
785                 return 0;
786         if (start == &unlocked)
```

```
787                 return 1;
788         return 0;
789 }

791 static int held_on_return(struct tracker *lock, struct smatch_state *start, enum
792 {
793         struct locks_on_return *tmp;

795         FOR_EACH_PTR(all_returns, tmp) {
796                 if (!matches_return_type(tmp->return_values, type))
797                         continue;
798                 if (match_held(lock, tmp, start))
799                         return 1;
800         } END_FOR_EACH_PTR(tmp);
801         return 0;
802 }

804 static int released_on_return(struct tracker *lock, struct smatch_state *start,
805 {
806         struct locks_on_return *tmp;

808         FOR_EACH_PTR(all_returns, tmp) {
809                 if (!matches_return_type(tmp->return_values, type))
810                         continue;
811                 if (match_released(lock, tmp, start))
812                         return 1;
813         } END_FOR_EACH_PTR(tmp);
814         return 0;
815 }

817 static void check_returns_consistently(struct tracker *lock,
818                                        struct smatch_state *start)
819 {
820         struct symbol *type;

822         if (!held_on_return(lock, start, ret_any) ||
823             !released_on_return(lock, start, ret_any))
824                 return;

826         if (held_on_return(lock, start, ret_zero) &&
827             !held_on_return(lock, start, ret_non_zero))
828                 return;

830         if (held_on_return(lock, start, ret_positive) &&
831             !held_on_return(lock, start, ret_zero))
832                 return;

834         if (held_on_return(lock, start, ret_positive) &&
835             !held_on_return(lock, start, ret_negative))
836                 return;

838         type = cur_func_return_type();
839         if (type && type->type == SYM_PTR) {
840                 if (held_on_return(lock, start, ret_non_zero) &&
841                     !held_on_return(lock, start, ret_zero))
842                         return;
843         }

845         print_inconsistent_returns(lock, start);
846 }

848 static void check_consistency(struct symbol *sym)
849 {
850         struct tracker *tmp;

852         FOR_EACH_PTR(starts_locked, tmp) {
```



```
853                 if (in_tracker_list(starts_unlocked, tmp->owner, tmp->name,
854                                     tmp->sym))
855                         sm_error("locking inconsistency.  We assume "
856                                  "'%s' is both locked and unlocked at the "
857                                  "start.",
858                                  tmp->name);
859         } END_FOR_EACH_PTR(tmp);

861         FOR_EACH_PTR(starts_locked, tmp) {
862                 check_returns_consistently(tmp, &locked);
863         } END_FOR_EACH_PTR(tmp);

865         FOR_EACH_PTR(starts_unlocked, tmp) {
866                 check_returns_consistently(tmp, &unlocked);
867         } END_FOR_EACH_PTR(tmp);
868 }

870 static void clear_lists(void)
871 {
872         struct locks_on_return *tmp;

874         func_has_transition = FALSE;

876         free_trackers_and_list(&starts_locked);
877         free_trackers_and_list(&starts_unlocked);

879         FOR_EACH_PTR(all_returns, tmp) {
880                 free_trackers_and_list(&tmp->locked);
881                 free_trackers_and_list(&tmp->unlocked);
882                 free(tmp);
883         } END_FOR_EACH_PTR(tmp);
884         __free_ptr_list((struct ptr_list **)&all_returns);
885 }

887 static void match_func_end(struct symbol *sym)
888 {
889         if (__inline_fn)
890                 return;

892         if (func_has_transition)
893                 check_consistency(sym);
894 }

896 static void match_after_func(struct symbol *sym)
897 {
898         if (__inline_fn)
899                 return;
900         clear_lists();
901 }

903 static void register_lock(int index)
904 {
905         struct lock_info *lock = &lock_table[index];
906         void *idx = INT_PTR(index);

908         if (lock->return_type == ret_non_zero) {
909                 return_implies_state(lock->function, valid_ptr_min, valid_ptr_ma
910                 return_implies_state(lock->function, 0, 0, &match_lock_failed, i
911         } else if (lock->return_type == ret_any && lock->arg == RETURN_VAL) {
912                 add_function_assign_hook(lock->function, &match_returns_locked,
913         } else if (lock->return_type == ret_any) {
914                 add_function_hook(lock->function, &match_lock_unlock, idx);
915         } else if (lock->return_type == ret_zero) {
916                 return_implies_state(lock->function, 0, 0, &match_lock_held, idx
917                 return_implies_state(lock->function, -4095, -1, &match_lock_fail
918         }
```

```
919 }

921 static void load_table(struct lock_info *_lock_table, int size)
922 {
923         int i;

925         lock_table = _lock_table;

927         for (i = 0; i < size; i++) {
928                 if (lock_table[i].action == LOCK)
929                         register_lock(i);
930                 else
931                         add_function_hook(lock_table[i].function, &match_lock_un
932         }
933 }

935 /* print_held_locks() is used in check_call_tree.c */
936 void print_held_locks(void)
937 {
938         struct stree *stree;
939         struct sm_state *sm;
940         int i = 0;

942         stree = __get_cur_stree();
943         FOR_EACH_MY_SM(my_id, stree, sm) {
944                 if (sm->state != &locked)
945                         continue;
946                 if (i++)
947                         sm_printf(" ");
948                 sm_printf("'%s'", sm->name);
949         } END_FOR_EACH_SM(sm);
950 }

952 void check_locking(int id)
953 {
954         my_id = id;

956         if (option_project == PROJ_WINE)
957                 load_table(wine_lock_table, ARRAY_SIZE(wine_lock_table));
958         else if (option_project == PROJ_KERNEL)
959                 load_table(kernel_lock_table, ARRAY_SIZE(kernel_lock_table));
960         else
961                 return;

963         add_unmatched_state_hook(my_id, &unmatched_state);
964         add_pre_merge_hook(my_id, &pre_merge_hook);
965         add_split_return_callback(match_return);
966         add_hook(&match_func_end, END_FUNC_HOOK);
967         add_hook(&match_after_func, AFTER_FUNC_HOOK);

969 }
```

```
**********************************************************
    1693 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_logical_instead_of_bitwise.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;

static int is_bitshift(struct expression *expr)
{
        expr = strip_expr(expr);

        if (expr->type != EXPR_BINOP)
                return 0;
        if (expr->op == SPECIAL_LEFTSHIFT)
                return 1;
        return 0;
}

static void match_logic(struct expression *expr)
{
        sval_t sval;

        if (expr->type != EXPR_LOGICAL)
                return;

        if (get_macro_name(expr->pos))
                return;

        if (!get_value(expr->right, &sval)) {
                if (!get_value(expr->left, &sval))
                        return;
        }

        if (sval.value == 0 || sval.value == 1)
                return;

        sm_warning("should this be a bitwise op?");
}

static void match_assign(struct expression *expr)
{
        struct expression *right;

        right = strip_expr(expr->right);
        if (right->type != EXPR_LOGICAL)
```



```
                return;
        if (is_bitshift(right->left) || is_bitshift(right->right))
                sm_warning("should this be a bitwise op?");
}

void check_logical_instead_of_bitwise(int id)
{
        my_id = id;

        add_hook(&match_logic, LOGIC_HOOK);
        add_hook(&match_assign, ASSIGNMENT_HOOK);
}
```

```
**********************************************************
    3650 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_macro_side_effects.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_expression_stacks.h"

static int my_id;

static struct string_list *ignored_macros;
static struct position old_pos;

static struct smatch_state *alloc_my_state(struct expression *expr)
{
        struct smatch_state *state;
        char *name;

        state = __alloc_smatch_state(0);
        expr = strip_expr(expr);
        name = expr_to_str(expr);
        state->name = alloc_sname(name);
        free_string(name);
        state->data = expr;
        return state;
}

static int defined_inside_macro(struct position macro_pos, struct expression *ex
{
        char *name;
        struct symbol *sym;
        int ret = 0;

        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                goto free;
        if (!sym->scope || !sym->scope->token)
                goto free;
        if (positions_eq(macro_pos, sym->scope->token->pos))
                ret = 1;
free:
        free_string(name);
        return ret;
}

static int affected_inside_macro_before(struct expression *expr)
```



```
{
        struct sm_state *sm;
        struct sm_state *tmp;
        struct expression *old_mod;

        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return 0;

        FOR_EACH_PTR(sm->possible, tmp) {
                old_mod = tmp->state->data;
                if (!old_mod)
                        continue;
                if (positions_eq(old_mod->pos, expr->pos))
                        return 1;
        } END_FOR_EACH_PTR(tmp);
        return 0;
}

static int is_ignored_macro(const char *macro)
{
        char *tmp;

        FOR_EACH_PTR(ignored_macros, tmp) {
                if (!strcmp(tmp, macro))
                        return 1;
        } END_FOR_EACH_PTR(tmp);
        return 0;
}

static void match_unop(struct expression *raw_expr)
{
        struct expression *expr;
        char *macro, *name;

        if (raw_expr->op != SPECIAL_INCREMENT && raw_expr->op != SPECIAL_DECREME
                return;

        macro = get_macro_name(raw_expr->pos);
        if (!macro)
                return;

        expr = strip_expr(raw_expr->unop);

        if (defined_inside_macro(expr->pos, expr))
                return;

        if (is_ignored_macro(macro))
                return;

        if (!affected_inside_macro_before(expr)) {
                set_state_expr(my_id, expr, alloc_my_state(expr));
                old_pos = expr->pos;
                return;
        }

        if (!positions_eq(old_pos, expr->pos))
                return;

        name = expr_to_str(raw_expr);
        sm_warning("side effect in macro '%s' doing '%s'",
                macro, name);
        free_string(name);
}

static void match_stmt(struct statement *stmt)
```

```
127 {
128         if (!positions_eq(old_pos, stmt->pos))
129                 old_pos.line = 0;
130 }

132 static void register_ignored_macros(void)
133 {
134         struct token *token;
135         char *macro;
136         char name[256];

138         snprintf(name, 256, "%s.ignore_side_effects", option_project_str);

140         token = get_tokens_file(name);
141         if (!token)
142                 return;
143         if (token_type(token) != TOKEN_STREAMBEGIN)
144                 return;
145         token = token->next;
146         while (token_type(token) != TOKEN_STREAMEND) {
147                 if (token_type(token) != TOKEN_IDENT)
148                         return;
149                 macro = alloc_string(show_ident(token->ident));
150                 add_ptr_list(&ignored_macros, macro);
151                 token = token->next;
152         }
153         clear_token_alloc();
154 }

156 void check_macro_side_effects(int id)
157 {
158         my_id = id;

160         if (!option_spammy)
161                 return;

163         add_hook(&match_unop, OP_HOOK);
164         add_hook(&match_stmt, STMT_HOOK);
165         register_ignored_macros();
166 }
```

```
**********************************************************
    2101 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_macros.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_inside(struct expression *expr, struct position pos)
{
        char *name;
        int matched = 0;

        if (positions_eq(expr->pos, pos))
                matched++;
        if (positions_eq(expr->unop->pos, pos))
                matched++;
        if (matched != 1)
                return;
        name = get_macro_name(pos);
        if (!name)
                return;
        sm_warning("the '%s' macro might need parens", name);
}

static void match_one_side(struct expression *expr, struct position pos, int op)
{
        char *name;
        int matched = 0;

        if ((op == '+' || op == '*' || op == '|' || op == '&') && expr->op == op
                return;
        if (positions_eq(expr->right->pos, pos))
                matched++;
        if (positions_eq(expr->left->pos, pos))
                matched++;
        if (matched != 1)
                return;
        name = get_macro_name(pos);
        if (!name)
                return;
        if (option_project == PROJ_WINE && !strcmp("BEGIN", name))
                return;
        sm_warning("the '%s' macro might need parens", name);
}

static void match_join(struct expression *expr)
```



```
{
        if (expr->left->type == EXPR_PREOP)
                match_inside(expr->left, expr->pos);
        if (expr->right->type == EXPR_POSTOP)
                match_inside(expr->right, expr->pos);

        if (expr->left->type == EXPR_BINOP)
                match_one_side(expr->left, expr->pos, expr->op);
        if (expr->right->type == EXPR_BINOP)
                match_one_side(expr->right, expr->pos, expr->op);
}

void check_macros(int id)
{
        my_id = id;
        add_hook(&match_join, BINOP_HOOK);
        add_hook(&match_join, LOGIC_HOOK);
}
```

```
**********************************************************
    8644 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_memcpy_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

struct limiter {
        int buf_arg;
        int limit_arg;
};
static struct limiter b0_l2 = {0, 2};
static struct limiter b1_l2 = {1, 2};

struct string_list *ignored_structs;

static int get_the_max(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;

        if (get_hard_max(expr, sval))
                return 1;
        if (!option_spammy)
                return 0;
        if (get_fuzzy_max(expr, sval))
                return 1;
        if (!get_user_rl(expr, &rl))
                return 0;
        *sval = rl_max(rl);
        return 1;
}

static int bytes_to_end_of_struct(struct expression *expr)
{
        struct expression *deref;
        struct symbol *type;
        int struct_bytes;
        int offset;

        if (expr->type == EXPR_PREOP && expr->op == '&')
                expr = strip_parens(expr->unop);
        else {
                type = get_type(expr);
                if (!type || type->type != SYM_ARRAY)
```



```
                        return 0;
        }
        if (expr->type != EXPR_DEREF || !expr->member)
                return 0;
        deref = expr->deref;
        if (deref->type == EXPR_PREOP && deref->op == '*')
                deref = deref->unop;
        struct_bytes = get_array_size_bytes_max(deref);
        if (struct_bytes <= 0) {
                type = get_type(expr->deref);
                struct_bytes = type_bytes(type);
        }
        offset = get_member_offset_from_deref(expr);
        if (offset <= 0)
                return 0;
        return struct_bytes - expr->member_offset;
}

static int size_of_union(struct expression *expr)
{
        struct symbol *type;

        if (expr->type != EXPR_PREOP || expr->op != '&')
                return 0;
        expr = strip_parens(expr->unop);
        if (expr->type != EXPR_DEREF || !expr->member)
                return 0;
        expr = expr->unop;
        type = get_type(expr);
        if (!type || type->type != SYM_UNION)
                return 0;
        return type_bytes(type);
}

static int is_likely_multiple(int has, int needed, struct expression *limit)
{
        sval_t mult;

        limit = strip_parens(limit);
        if (limit->type != EXPR_BINOP || limit->op != '*')
                return 0;
        if (!get_value(limit->left, &mult))
                return 0;
        if (has * mult.value == needed)
                return 1;
        if (!get_value(limit->right, &mult))
                return 0;
        if (has * mult.value == needed)
                return 1;

        return 0;
}

static int name_in_union(struct symbol *type, const char *name)
{
        struct symbol *tmp;

        if (type->type == SYM_NODE)
                type = get_real_base_type(type);
        if (!type || type->type != SYM_UNION)
                return 0;

        FOR_EACH_PTR(type->symbol_list, tmp) {
                if (tmp->ident &&
                    strcmp(name, tmp->ident->name) == 0)
                        return 1;
```

```
127         } END_FOR_EACH_PTR(tmp);

129         return 0;
130 }

132 static int ends_on_struct_member_boundary(struct expression *expr, int needed)
133 {
134         struct symbol *type, *tmp;
135         int offset;
136         int size;
137         int found = 0;

139         expr = strip_expr(expr);
140         if (expr->type == EXPR_PREOP && expr->op == '&') {
141                 expr = strip_parens(expr->unop);
142         } else {
143                 type = get_type(expr);
144                 if (!type || type->type != SYM_ARRAY)
145                         return 0;
146         }
147         if (expr->type != EXPR_DEREF || !expr->member)
148                 return 0;

150         type = get_type(expr->unop);
151         if (!type)
152                 return 0;
153         if (type->type == SYM_UNION) {
154                 struct expression *unop = strip_expr(expr->unop);

156                 if (unop->type != EXPR_DEREF)
157                         return 0;
158                 type = get_type(unop->unop);
159                 if (!type)
160                         return 0;
161         }
162         if (type->type != SYM_STRUCT)
163                 return 0;

165         offset = 0;
166         FOR_EACH_PTR(type->symbol_list, tmp) {
167                 if (!found) {
168                         if ((tmp->ident &&
169                              strcmp(expr->member->name, tmp->ident->name) == 0)
170                             name_in_union(tmp, expr->member->name))
171                                 found = 1;

173                         offset = ALIGN(offset, tmp->ctype.alignment);

175                         offset += type_bytes(tmp);
176                         size = type_bytes(tmp);
177                         continue;
178                 }

180                 /* if there is a hole then fail. */
181                 if (offset != ALIGN(offset, tmp->ctype.alignment))
182                         return 0;
183                 offset += type_bytes(tmp);
184                 size += type_bytes(tmp);

186                 if (size == needed)
187                         return 1;
188                 if (size > needed)
189                         return 0;
190         } END_FOR_EACH_PTR(tmp);
191         return 0;
192 }
```



```
194 static int is_one_element_array(struct expression *expr)
195 {
196         struct symbol *type;
197         sval_t sval;

199         if (expr->type == EXPR_PREOP && expr->op == '&')
200                 expr = expr->unop;
201         if (expr->type == EXPR_BINOP) /* array elements foo[5] */
202                 return 0;

204         type = get_type(expr);
205         if (!type)
206                 return 0;
207         if (!type || type->type != SYM_ARRAY)
208                 return 0;

210         if (!get_implied_value(type->array_size, &sval))
211                 return 0;

213         if (sval.value == 1)
214                 return 1;
215         return 0;
216 }

218 static int is_ignored_struct(struct expression *expr)
219 {
220         struct symbol *type;

222         type = get_type(expr);
223         if (!type)
224                 return 0;
225         if (type->type == SYM_PTR)
226                 type = get_real_base_type(type);
227         if (type->type != SYM_STRUCT)
228                 return 0;
229         if (!type->ident)
230                 return 0;
231         if (list_has_string(ignored_structs, type->ident->name))
232                 return 1;
233         return 0;
234 }

236 static void match_limited(const char *fn, struct expression *expr, void *_limite
237 {
238         struct limiter *limiter = (struct limiter *)_limiter;
239         struct expression *dest;
240         struct expression *limit;
241         char *dest_name = NULL;
242         sval_t needed;
243         int has;

245         dest = get_argument_from_call_expr(expr->args, limiter->buf_arg);
246         limit = get_argument_from_call_expr(expr->args, limiter->limit_arg);
247         if (!get_the_max(limit, &needed))
248                 return;
249         has = get_array_size_bytes_max(dest);
250         if (!has)
251                 return;
252         if (has >= needed.value)
253                 return;

255         if (needed.value == bytes_to_end_of_struct(dest))
256                 return;

258         if (needed.value <= size_of_union(dest))
```

```
259                 return;

261         if (is_likely_multiple(has, needed.value, limit))
262                 return;

264         if (ends_on_struct_member_boundary(dest, needed.value))
265                 return;

267         if (is_one_element_array(dest))
268                 return;

270         if (is_ignored_struct(dest))
271                 return;

273         dest_name = expr_to_str(dest);
274         sm_error("%s() '%s' too small (%d vs %s)", fn, dest_name, has, sval_to_s
275         free_string(dest_name);
276 }

278 static void register_funcs_from_file(void)
279 {
280         char name[256];
281         struct token *token;
282         const char *func;
283         int size, buf;
284         struct limiter *limiter;

286         snprintf(name, 256, "%s.sizeof_param", option_project_str);
287         name[255] = '\0';
288         token = get_tokens_file(name);
289         if (!token)
290                 return;
291         if (token_type(token) != TOKEN_STREAMBEGIN)
292                 return;
293         token = token->next;
294         while (token_type(token) != TOKEN_STREAMEND) {
295                 if (token_type(token) != TOKEN_IDENT)
296                         break;
297                 func = show_ident(token->ident);

299                 token = token->next;
300                 if (token_type(token) != TOKEN_NUMBER)
301                         break;
302                 size = atoi(token->number);

304                 token = token->next;
305                 if (token_type(token) == TOKEN_SPECIAL) {
306                         if (token->special != '-')
307                                 break;
308                         token = token->next;
309                         if (token_type(token) != TOKEN_NUMBER)
310                                 break;
311                         token = token->next;
312                         continue;

314                 }
315                 if (token_type(token) != TOKEN_NUMBER)
316                         break;
317                 buf = atoi(token->number);

319                 limiter = malloc(sizeof(*limiter));
320                 limiter->limit_arg = size;
321                 limiter->buf_arg = buf;

323                 add_function_hook(func, &match_limited, limiter);
```



```
325                 token = token->next;
326         }
327         if (token_type(token) != TOKEN_STREAMEND)
328                 sm_perror("parsing '%s'", name);
329         clear_token_alloc();
330 }

332 static void register_ignored_structs_from_file(void)
333 {
334         char name[256];
335         struct token *token;
336         const char *struct_type;

338         snprintf(name, 256, "%s.ignore_memcpy_struct_overflows", option_project_
339         name[255] = '\0';
340         token = get_tokens_file(name);
341         if (!token)
342                 return;
343         if (token_type(token) != TOKEN_STREAMBEGIN)
344                 return;
345         token = token->next;
346         while (token_type(token) != TOKEN_STREAMEND) {
347                 if (token_type(token) != TOKEN_IDENT)
348                         return;

350                 struct_type = show_ident(token->ident);
351                 insert_string(&ignored_structs, alloc_string(struct_type));

353                 token = token->next;
354         }
355         clear_token_alloc();
356 }

358 void check_memcpy_overflow(int id)
359 {
360         register_funcs_from_file();
361         register_ignored_structs_from_file();
362         add_function_hook("memcmp", &match_limited, &b0_l2);
363         add_function_hook("memcmp", &match_limited, &b1_l2);
364         if (option_project == PROJ_KERNEL) {
365                 add_function_hook("copy_to_user", &match_limited, &b1_l2);
366                 add_function_hook("_copy_to_user", &match_limited, &b1_l2);
367                 add_function_hook("__copy_to_user", &match_limited, &b1_l2);
368                 add_function_hook("copy_from_user", &match_limited, &b0_l2);
369                 add_function_hook("_copy_from_user", &match_limited, &b0_l2);
370                 add_function_hook("__copy_from_user", &match_limited, &b0_l2);
371         }
372 }
```

```
**********************************************************
   10048 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_memory.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2008 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <fcntl.h>
#include <unistd.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"

static void check_sm_is_leaked(struct sm_state *sm);

static int my_id;

STATE(allocated);
STATE(assigned);
STATE(isfree);
STATE(malloced);
STATE(isnull);
STATE(unfree);

/*
  malloced --> allocated --> assigned --> isfree +
               \-> isnull.    \-> isfree +

  isfree --> unfree.
             \-> isnull.
*/

static struct tracker_list *arguments;

static const char *allocation_funcs[] = {
        "malloc",
        "kmalloc",
        "kzalloc",
        NULL,
};

static char *get_parent_name(struct symbol *sym)
{
        static char buf[256];

        if (!sym || !sym->ident)
                return NULL;

        snprintf(buf, 255, "-%s", sym->ident->name);
        buf[255] = '\0';
```



```
        return alloc_string(buf);
}

static int is_parent_sym(const char *name)
{
        if (!strncmp(name, "-", 1))
                return 1;
        return 0;
}

static int is_complex(struct expression *expr)
{
        char *name;
        int ret = 1;

        name = expr_to_var(expr);
        if (name)
                ret = 0;
        free_string(name);
        return ret;
}

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        if (is_parent_sym(sm->name))
                return &assigned;
        return &undefined;
}

static void assign_parent(struct symbol *sym)
{
        char *name;

        name = get_parent_name(sym);
        if (!name)
                return;
        set_state(my_id, name, sym, &assigned);
        free_string(name);
}

static int parent_is_assigned(struct symbol *sym)
{
        struct smatch_state *state;
        char *name;

        name = get_parent_name(sym);
        if (!name)
                return 0;
        state = get_state(my_id, name, sym);
        free_string(name);
        if (state == &assigned)
                return 1;
        return 0;
}

static int is_allocation(struct expression *expr)
{
        char *fn_name;
        int i;

        if (expr->type != EXPR_CALL)
                return 0;

        if (!(fn_name = expr_to_var_sym(expr->fn, NULL)))
                return 0;
```

```
127         for (i = 0; allocation_funcs[i]; i++) {
128                 if (!strcmp(fn_name, allocation_funcs[i])) {
129                         free_string(fn_name);
130                         return 1;
131                 }
132         }
133         free_string(fn_name);
134         return 0;
135 }

137 static int is_freed(const char *name, struct symbol *sym)
138 {
139         struct state_list *slist;

141         slist = get_possible_states(my_id, name, sym);
142         if (slist_has_state(slist, &isfree)) {
143                 return 1;
144         }
145         return 0;
146 }

148 static int is_argument(struct symbol *sym)
149 {
150         struct tracker *arg;

152         FOR_EACH_PTR(arguments, arg) {
153                 if (arg->sym == sym)
154                         return 1;
155         } END_FOR_EACH_PTR(arg);
156         return 0;
157 }

159 static void match_function_def(struct symbol *sym)
160 {
161         struct symbol *arg;

163         FOR_EACH_PTR(sym->ctype.base_type->arguments, arg) {
164                 add_tracker(&arguments, my_id, (arg->ident?arg->ident->name:"NUL
165         } END_FOR_EACH_PTR(arg);
166 }

168 static int is_parent(struct expression *expr)
169 {
170         if (expr->type == EXPR_DEREF)
171                 return 0;
172         return 1;
173 }

175 static void match_assign(struct expression *expr)
176 {
177         struct expression *left, *right;
178         char *left_name = NULL;
179         char *right_name = NULL;
180         struct symbol *left_sym, *right_sym;
181         struct smatch_state *state;
182         struct state_list *slist;
183         struct sm_state *tmp;

185         left = strip_expr(expr->left);
186         left_name = expr_to_str_sym(left, &left_sym);

188         right = strip_expr(expr->right);
189         while (right->type == EXPR_ASSIGNMENT)
190                 right = right->left;

192         if (left_name && left_sym && is_allocation(right) &&
```



```
193             !(left_sym->ctype.modifiers &
194               (MOD_NONLOCAL | MOD_STATIC | MOD_ADDRESSABLE)) &&
195             !parent_is_assigned(left_sym)) {
196                 set_state(my_id, left_name, left_sym, &malloced);
197                 goto exit;
198         }

200         right_name = expr_to_str_sym(right, &right_sym);

202         if (right_name && (state = get_state(my_id, right_name, right_sym))) {
203                 if (state == &isfree && !is_complex(right))
204                         sm_error("assigning freed pointer '%s'", right_name);
205                 set_state(my_id, right_name, right_sym, &assigned);
206         }

208         if (is_zero(expr->right)) {
209                 slist = get_possible_states(my_id, left_name, left_sym);

211                 FOR_EACH_PTR(slist, tmp) {
212                         check_sm_is_leaked(tmp);
213                 } END_FOR_EACH_PTR(tmp);
214         }

216         if (is_freed(left_name, left_sym)) {
217                 set_state(my_id, left_name, left_sym, &unfree);
218         }
219         if (left_name && is_parent(left))
220                 assign_parent(left_sym);
221         if (right_name && is_parent(right))
222                 assign_parent(right_sym);
223 exit:
224         free_string(left_name);
225         free_string(right_name);
226 }

228 static int is_null(const char *name, struct symbol *sym)
229 {
230         struct smatch_state *state;

232         state = get_state(my_id, name, sym);
233         if (state && !strcmp(state->name, "isnull"))
234                 return 1;
235         return 0;
236 }

238 static void set_unfree(struct sm_state *sm, struct expression *mod_expr)
239 {
240         if (slist_has_state(sm->possible, &isfree))
241                 set_state(my_id, sm->name, sm->sym, &unfree);
242 }

244 static void match_free_func(const char *fn, struct expression *expr, void *data)
245 {
246         struct expression *ptr_expr;
247         char *ptr_name;
248         struct symbol *ptr_sym;
249         int arg_num = PTR_INT(data);

251         ptr_expr = get_argument_from_call_expr(expr->args, arg_num);
252         ptr_name = expr_to_var_sym(ptr_expr, &ptr_sym);
253         if (!ptr_name)
254                 return;
255         set_state(my_id, ptr_name, ptr_sym, &isfree);
256         free_string(ptr_name);
257 }
```

```
259 static int possibly_allocated(struct state_list *slist)
260 {
261         struct sm_state *tmp;

263         FOR_EACH_PTR(slist, tmp) {
264                 if (tmp->state == &allocated)
265                         return 1;
266                 if (tmp->state == &malloced)
267                         return 1;
268         } END_FOR_EACH_PTR(tmp);
269         return 0;
270 }

272 static void check_sm_is_leaked(struct sm_state *sm)
273 {
274         if (possibly_allocated(sm->possible) &&
275                 !is_null(sm->name, sm->sym) &&
276                 !is_argument(sm->sym) &&
277                 !parent_is_assigned(sm->sym))
278                 sm_error("memory leak of '%s'", sm->name);
279 }

281 static void check_tracker_is_leaked(struct tracker *t)
282 {
283         struct sm_state *sm;

285         sm = get_sm_state(t->owner, t->name, t->sym);
286         if (sm)
287                 check_sm_is_leaked(sm);
288         __free_tracker(t);
289 }

291 static void match_declarations(struct symbol *sym)
292 {
293         const char *name;

295         if ((get_base_type(sym))->type == SYM_ARRAY) {
296                 return;
297         }

299         name = sym->ident->name;

301         if (sym->initializer) {
302                 if (is_allocation(sym->initializer)) {
303                         set_state(my_id, name, sym, &malloced);
304                         add_scope_hook((scope_hook *)&check_tracker_is_leaked,
305                                 alloc_tracker(my_id, name, sym));
306                         scoped_state(my_id, name, sym);
307                 } else {
308                         assign_parent(sym);
309                 }
310         }
311 }

313 static void check_for_allocated(void)
314 {
315         struct stree *stree;
316         struct sm_state *tmp;

318         stree = __get_cur_stree();
319         FOR_EACH_MY_SM(my_id, stree, tmp) {
320                 check_sm_is_leaked(tmp);
321         } END_FOR_EACH_SM(tmp);
322 }

324 static void match_return(struct expression *ret_value)
```



```
325 {
326         char *name;
327         struct symbol *sym;

329         if (__inline_fn)
330                 return;
331         name = expr_to_str_sym(ret_value, &sym);
332         if (sym)
333                 assign_parent(sym);
334         free_string(name);
335         check_for_allocated();
336 }

338 static void set_new_true_false_paths(const char *name, struct symbol *sym)
339 {
340         struct smatch_state *tmp;

342         tmp = get_state(my_id, name, sym);

344         if (!tmp) {
345                 return;
346         }

348         if (tmp == &isfree) {
349                 sm_warning("why do you care about freed memory? '%s'", name);
350         }

352         if (tmp == &assigned) {
353                 /* we don't care about assigned pointers any more */
354                 return;
355         }
356         set_true_false_states(my_id, name, sym, &allocated, &isnull);
357 }

359 static void match_condition(struct expression *expr)
360 {
361         struct symbol *sym;
362         char *name;

364         expr = strip_expr(expr);
365         switch (expr->type) {
366         case EXPR_PREOP:
367         case EXPR_SYMBOL:
368         case EXPR_DEREF:
369                 name = expr_to_var_sym(expr, &sym);
370                 if (!name)
371                         return;
372                 set_new_true_false_paths(name, sym);
373                 free_string(name);
374                 return;
375         case EXPR_ASSIGNMENT:
376                  /* You have to deal with stuff like if (a = b = c) */
377                 match_condition(expr->right);
378                 match_condition(expr->left);
379                 return;
380         default:
381                 return;
382         }
383 }

385 static void match_function_call(struct expression *expr)
386 {
387         struct expression *tmp;
388         struct symbol *sym;
389         char *name;
390         struct sm_state *state;
```

```
392         FOR_EACH_PTR(expr->args, tmp) {
393                 tmp = strip_expr(tmp);
394                 name = expr_to_str_sym(tmp, &sym);
395                 if (!name)
396                         continue;
397                 if ((state = get_sm_state(my_id, name, sym))) {
398                         if (possibly_allocated(state->possible)) {
399                                 set_state(my_id, name, sym, &assigned);
400                         }
401                 }
402                 assign_parent(sym);
403                 free_string(name);
404         } END_FOR_EACH_PTR(tmp);
405 }

407 static void match_end_func(struct symbol *sym)
408 {
409         if (__inline_fn)
410                 return;
411         check_for_allocated();
412 }

414 static void match_after_func(struct symbol *sym)
415 {
416         if (__inline_fn)
417                 return;
418         free_trackers_and_list(&arguments);
419 }

421 static void register_funcs_from_file(void)
422 {
423         struct token *token;
424         const char *func;
425         int arg;

427         token = get_tokens_file("kernel.frees_argument");
428         if (!token)
429                 return;
430         if (token_type(token) != TOKEN_STREAMBEGIN)
431                 return;
432         token = token->next;
433         while (token_type(token) != TOKEN_STREAMEND) {
434                 if (token_type(token) != TOKEN_IDENT)
435                         return;
436                 func = show_ident(token->ident);
437                 token = token->next;
438                 if (token_type(token) != TOKEN_NUMBER)
439                         return;
440                 arg = atoi(token->number);
441                 add_function_hook(func, &match_free_func, INT_PTR(arg));
442                 token = token->next;
443         }
444         clear_token_alloc();
445 }

447 void check_memory(int id)
448 {
449         my_id = id;
450         add_unmatched_state_hook(my_id, &unmatched_state);
451         add_hook(&match_function_def, FUNC_DEF_HOOK);
452         add_hook(&match_declarations, DECLARATION_HOOK);
453         add_hook(&match_function_call, FUNCTION_CALL_HOOK);
454         add_hook(&match_condition, CONDITION_HOOK);
455         add_hook(&match_assign, ASSIGNMENT_HOOK);
456         add_hook(&match_return, RETURN_HOOK);
```



```
457         add_hook(&match_end_func, END_FUNC_HOOK);
458         add_hook(&match_after_func, AFTER_FUNC_HOOK);
459         add_modification_hook(my_id, &set_unfree);
460         if (option_project == PROJ_KERNEL) {
461                 add_function_hook("kfree", &match_free_func, (void *)0);
462                 register_funcs_from_file();
463         } else {
464                 add_function_hook("free", &match_free_func, (void *)0);
465         }
466 }
```

```
**********************************************************
    1237 Fri Dec 21 15:00:05 2018
new/usr/src/tools/smatch/src/check_memset.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_memset(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg_expr;
        sval_t sval;

        arg_expr = get_argument_from_call_expr(expr->args, 2);

        if (arg_expr->type != EXPR_VALUE)
                return;
        if (!get_value(arg_expr, &sval))
                return;
        if (sval.value != 0)
                return;
        sm_error("calling memset(x, y, 0);");
}

void check_memset(int id)
{
        my_id = id;
        add_function_hook("memset", &match_memset, NULL);
        add_function_hook("__builtin_memset", &match_memset, NULL);
}
```

```
**********************************************************
    1447 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_min_t.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_assign(struct expression *expr)
{
        const char *macro;
        sval_t max_left, max_right;
        char *name;

        if (expr->op != '=')
                return;

        macro = get_macro_name(expr->pos);
        if (!macro)
                return;
        if (strcmp(macro, "min_t"))
                return;

        if (!get_absolute_max(expr->left, &max_left))
                return;
        if (!get_absolute_max(expr->right, &max_right))
                return;

        if (sval_cmp(max_left, max_right) >= 0)
                return;

        name = expr_to_str(expr->right);
        sm_warning("min_t truncates here '%s' (%s vs %s)", name, sval_to_str(max
        free_string(name);
}

void check_min_t(int id)
{
        my_id = id;
        if (option_project != PROJ_KERNEL)
                return;
        add_hook(&match_assign, ASSIGNMENT_HOOK);
}
```

```
**********************************************************
    4285 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_missing_break.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The way I'm detecting missing breaks is if there is an assignment inside a
 * switch statement which is over written.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;
static struct expression *skip_this;

/*
 * It goes like this:
 * - Allocate a state which stores the switch expression.  I wanted to
 *   just have a state &assigned but we need to know the switch statement where
 *   it was assigned.
 * - If it gets used then we change it to &used.
 * - For unmatched states we use &used (because of cleanness, not because we nee
 *   to).
 * - If we merge inside a case statement and one of the states is &assigned (or
 *   if it is &nobreak) then &nobreak is used.
 *
 * We print an error when we assign something to a &no_break symbol.
 *
 */

STATE(used);
STATE(no_break);

static int in_switch_stmt;

static struct smatch_state *alloc_my_state(struct expression *expr)
{
        struct smatch_state *state;
        char *name;

        state = __alloc_smatch_state(0);
        expr = strip_expr(expr);
        name = expr_to_str(expr);
        if (!name)
                name = alloc_string("");
        state->name = alloc_sname(name);
        free_string(name);
        state->data = expr;
        return state;
}

struct expression *last_print_expr;
static void print_missing_break(struct expression *expr)
{
        char *name;

        if (get_switch_expr() == last_print_expr)
                return;
        last_print_expr = get_switch_expr();

        name = expr_to_var(expr);
        sm_warning("missing break? reassigning '%s'", name);
        free_string(name);
}

static void match_assign(struct expression *expr)
{
        struct expression *left;

        if (expr->op != '=')
                return;
        if (!get_switch_expr())
                return;
        left = strip_expr(expr->left);
        if (get_state_expr(my_id, left) == &no_break)
                print_missing_break(left);

        set_state_expr(my_id, left, alloc_my_state(get_switch_expr()));
        skip_this = left;
}

static void match_symbol(struct expression *expr)
{
        if (outside_of_function())
                return;
        if (!get_switch_expr())
                return;

        expr = strip_expr(expr);
        if (expr == skip_this)
                return;
        set_state_expr(my_id, expr, &used);
}

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        return &used;
}

static int in_case;
static struct smatch_state *merge_hook(struct smatch_state *s1, struct smatch_st
{
        struct expression *switch_expr;

        if (s1 == &no_break || s2 == &no_break)
                return &no_break;
        if (!in_case)
                return &used;
        switch_expr = get_switch_expr();
        if (s1->data == switch_expr || s2->data == switch_expr)
                return &no_break;
        return &used;
```

```
127 }

129 static void match_stmt(struct statement *stmt)
130 {
131         if (stmt->type == STMT_CASE)
132                 in_case = 1;
133         else
134                 in_case = 0;
135 }

137 static void match_switch(struct statement *stmt)
138 {
139         if (stmt->type != STMT_SWITCH)
140                 return;

142         in_switch_stmt++;
143 }

145 static void delete_my_states(int owner)
146 {
147         struct state_list *slist = NULL;
148         struct sm_state *sm;

150         FOR_EACH_MY_SM(owner, __get_cur_stree(), sm) {
151                 add_ptr_list(&slist, sm);
152         } END_FOR_EACH_SM(sm);

154         FOR_EACH_PTR(slist, sm) {
155                 delete_state(sm->owner, sm->name, sm->sym);
156         } END_FOR_EACH_PTR(sm);

158         free_slist(&slist);
159 }

161 static void match_switch_end(struct statement *stmt)
162 {

164         if (stmt->type != STMT_SWITCH)
165                 return;

167         in_switch_stmt--;

169         if (!in_switch_stmt)
170                 delete_my_states(my_id);
171 }

173 void check_missing_break(int id)
174 {
175         my_id = id;

177         if (!option_spammy)
178                 return;

180         add_unmatched_state_hook(my_id, &unmatched_state);
181         add_merge_hook(my_id, &merge_hook);

183         add_hook(&match_assign, ASSIGNMENT_HOOK);
184         add_hook(&match_symbol, SYM_HOOK);
185         add_hook(&match_stmt, STMT_HOOK);
186         add_hook(&match_switch, STMT_HOOK);
187         add_hook(&match_switch_end, STMT_HOOK_AFTER);
188 }
```

**********
1215 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_mod_timer.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static void match_mod_timer(const char *fn, struct expression *expr, void *param
{
        struct expression *arg;
        sval_t sval;

        arg = get_argument_from_call_expr(expr->args, 1);
        if (!get_value(arg, &sval) || sval.value == 0)
                return;
        sm_warning("mod_timer() takes an absolute time not an offset.");
}

void check_mod_timer(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        my_id = id;
        add_function_hook("mod_timer", &match_mod_timer, NULL);
}
```

```
**********************************************************
    1389 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_no_effect.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_stmt(struct statement *stmt)
{
        struct expression *expr;

        if (stmt->type != STMT_EXPRESSION)
                return;
        expr = stmt->expression;
        if (!expr)
                return;
        switch(expr->type) {
        case EXPR_PREOP:
                if (expr->op == '!')
                        break;
                if (expr->op == '~')
                        break;
        case EXPR_POSTOP:
        case EXPR_STATEMENT:
        case EXPR_ASSIGNMENT:
        case EXPR_CALL:
        case EXPR_CONDITIONAL:
        case EXPR_SELECT:
        case EXPR_CAST:
        case EXPR_FORCE_CAST:
        case EXPR_COMMA:
                return;
        }
        if (in_expression_statement())
                return;
        sm_warning("statement has no effect %d", expr->type);
}

void check_no_effect(int id)
{
        my_id = id;
        add_hook(&match_stmt, STMT_HOOK);
}
```

```
**********************************************************
    1514 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_no_if_block.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_if_stmt(struct statement *stmt)
{
        if (__inline_fn)
                return;
        if (stmt->type != STMT_IF)
                return;
        if (stmt->if_true->type == STMT_COMPOUND)
                return;
        if (get_macro_name(stmt->pos))
                return;
        if (stmt->pos.pos != stmt->if_true->pos.pos)
                return;
        sm_warning("if statement not indented");
}

static void match_for_stmt(struct statement *stmt)
{
        if (__inline_fn)
                return;
        if (stmt->type != STMT_ITERATOR)
                return;
        if (stmt->iterator_statement->type == STMT_COMPOUND)
                return;
        if (get_macro_name(stmt->pos))
                return;
        if (stmt->pos.pos != stmt->iterator_statement->pos.pos)
                return;
        sm_warning("for statement not indented");
}

void check_no_if_block(int id)
{
        my_id = id;

        add_hook(&match_if_stmt, STMT_HOOK);
        add_hook(&match_for_stmt, STMT_HOOK);
}
```

```
**********************************************************
    1213 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_no_return.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;
static int returned;

static void match_return(struct expression *ret_value)
{
        if (__inline_fn)
                return;
        if (is_reachable())
                returned = 1;
}

static void match_func_end(struct symbol *sym)
{
        if (__inline_fn)
                return;
        if (!is_reachable() && !returned)
                sm_info("info: add to no_return_funcs");
        returned = 0;
}

void check_no_return(int id)
{
        if (!option_info)
                return;
        my_id = id;
        add_hook(&match_return, RETURN_HOOK);
        add_hook(&match_func_end, END_FUNC_HOOK);
}
```

```
**********************************************************
    6280 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_nospec.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2018 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;
static int barrier_id;

STATE(nospec);

static bool in_nospec_stmt;

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        struct range_list *rl;

        if (__in_function_def && !get_user_rl_var_sym(sm->name, sm->sym, &rl))
                return &nospec;
        return &undefined;
}

bool is_nospec(struct expression *expr)
{
        char *macro;

        if (in_nospec_stmt)
                return true;
        if (!expr)
                return false;
        if (get_state_expr(my_id, expr) == &nospec)
                return true;
        macro = get_macro_name(expr->pos);
        if (macro && strcmp(macro, "array_index_nospec") == 0)
                return true;
        return false;
}

static void nospec_assign(struct expression *expr)
{
        if (is_nospec(expr->right))
                set_state_expr(my_id, expr->left, &nospec);
}
```



```
static void set_param_nospec(const char *name, struct symbol *sym, char *key, ch
{
        char fullname[256];

        if (strcmp(key, "*$") == 0)
                snprintf(fullname, sizeof(fullname), "*%s", name);
        else if (strncmp(key, "$", 1) == 0)
                snprintf(fullname, 256, "%s%s", name, key + 1);
        else
                return;

        set_state(my_id, fullname, sym, &nospec);
}

static void match_call_info(struct expression *expr)
{
        struct expression *arg;
        int i = 0;

        FOR_EACH_PTR(expr->args, arg) {
                if (get_state_expr(my_id, arg) == &nospec)
                        sql_insert_caller_info(expr, NOSPEC, i, "$", "");
                i++;
        } END_FOR_EACH_PTR(arg);
}

static void struct_member_callback(struct expression *call, int param, char *pri
{
        struct range_list *rl;

        if (!get_user_rl_var_sym(sm->name, sm->sym, &rl))
                return;
        sql_insert_caller_info(call, NOSPEC, param, printed_name, "");
}

static void returned_struct_members(int return_id, char *return_ranges, struct e
{
        struct symbol *returned_sym;
        struct sm_state *sm;
        const char *param_name;
        struct range_list *rl;
        int param;

        returned_sym = expr_to_sym(expr);

        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
                param = get_param_num_from_sym(sm->sym);
                if (param < 0) {
                        if (!returned_sym || returned_sym != sm->sym)
                                continue;
                        param = -1;
                }

                param_name = get_param_name(sm);
                if (!param_name)
                        continue;
                if (param != -1 && strcmp(param_name, "$") == 0)
                        continue;

                if (!get_user_rl_var_sym(sm->name, sm->sym, &rl))
                        continue;

                sql_insert_return_states(return_id, return_ranges, NOSPEC, param
        } END_FOR_EACH_SM(sm);
```

```
127         if (is_nospec(expr) && get_user_rl(expr, &rl))
128                 sql_insert_return_states(return_id, return_ranges, NOSPEC, -1, "

130         if (get_state(barrier_id, "barrier", NULL) == &nospec)
131                 sql_insert_return_states(return_id, return_ranges, NOSPEC_WB, -1
132 }

134 static int is_return_statement(void)
135 {
136         if (__cur_stmt && __cur_stmt->type == STMT_RETURN)
137                 return 1;
138         return 0;
139 }

141 static void db_returns_nospec(struct expression *expr, int param, char *key, cha
142 {
143         struct expression *call;
144         struct expression *arg;
145         char *name;
146         struct symbol *sym;

148         call = expr;
149         while (call->type == EXPR_ASSIGNMENT)
150                 call = strip_expr(call->right);
151         if (call->type != EXPR_CALL)
152                 return;

154         if (param == -1 && expr->type == EXPR_ASSIGNMENT) {
155                 name = get_variable_from_key(expr->left, key, &sym);
156         } else if (param == -1 && is_return_statement()) {
157                 in_nospec_stmt = true;
158                 return;
159         } else {
160                 arg = get_argument_from_call_expr(call->args, param);
161                 if (!arg)
162                         return;
163                 name = get_variable_from_key(arg, key, &sym);
164         }
165         if (!name || !sym)
166                 goto free;

168         set_state(my_id, name, sym, &nospec);
169 free:
170         free_string(name);
171 }

173 static int is_nospec_asm(struct statement *stmt)
174 {
175         char *macro;

177         if (!stmt || stmt->type != STMT_ASM)
178                 return 0;
179         macro = get_macro_name(stmt->asm_string->pos);
180         if (!macro || strcmp(macro, "CALL_NOSPEC") != 0)
181                 return 0;
182         return 1;
183 }

185 static void match_asm(struct statement *stmt)
186 {
187         if (is_nospec_asm(stmt))
188                 in_nospec_stmt = true;
189 }

191 static void match_after_nospec_asm(struct statement *stmt)
192 {
```



```
193         in_nospec_stmt = false;
194 }

196 static void mark_user_data_as_nospec(void)
197 {
198         struct stree *stree;
199         struct symbol *type;
200         struct sm_state *sm;

202         stree = get_user_stree();
203         FOR_EACH_SM(stree, sm) {
204                 if (is_whole_rl(estate_rl(sm->state)))
205                         continue;
206                 type = estate_type(sm->state);
207                 if (!type || type->type != SYM_BASETYPE)
208                         continue;
209                 if (!is_capped_var_sym(sm->name, sm->sym))
210                         continue;
211                 set_state(my_id, sm->name, sm->sym, &nospec);
212         } END_FOR_EACH_SM(sm);
213         free_stree(&stree);
214 }

216 static void match_barrier(struct statement *stmt)
217 {
218         char *macro;

220         macro = get_macro_name(stmt->pos);
221         if (!macro)
222                 return;
223         if (strcmp(macro, "rmb") != 0 &&
224             strcmp(macro, "smp_rmb") != 0 &&
225             strcmp(macro, "barrier_nospec") != 0)
226                 return;

228         set_state(barrier_id, "barrier", NULL, &nospec);
229         mark_user_data_as_nospec();
230 }

232 static void db_returns_barrier(struct expression *expr, int param, char *key, ch
233 {
234         mark_user_data_as_nospec();
235 }

237 void check_nospec(int id)
238 {
239         my_id = id;

241         add_hook(&nospec_assign, ASSIGNMENT_HOOK);

243         select_caller_info_hook(set_param_nospec, NOSPEC);
244         add_unmatched_state_hook(my_id, &unmatched_state);

246         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
247         add_member_info_callback(my_id, struct_member_callback);
248         add_split_return_callback(&returned_struct_members);
249         select_return_states_hook(NOSPEC, &db_returns_nospec);
250         select_return_states_hook(NOSPEC_WB, &db_returns_barrier);

252         add_hook(&match_asm, ASM_HOOK);
253         add_hook(&match_after_nospec_asm, STMT_HOOK_AFTER);
254 }

256 void check_nospec_barrier(int id)
257 {
258         barrier_id = id;
```

```
260         add_hook(&match_barrier, ASM_HOOK);
261 }
```

```
**********************************************************
    3689 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_off_by_one_relative.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The point here is to store that a buffer has x bytes even if we don't know
 * the value of x.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static void array_check(struct expression *expr)
{
        struct expression *array;
        struct expression *size;
        struct expression *offset;
        char *array_str, *offset_str;

        expr = strip_expr(expr);
        if (!is_array(expr))
                return;

        array = get_array_base(expr);
        size = get_size_variable(array);
        if (!size)
                return;
        offset = get_array_offset(expr);
        if (!possible_comparison(size, SPECIAL_EQUAL, offset))
                return;

        array_str = expr_to_str(array);
        offset_str = expr_to_str(offset);
        sm_warning("potentially one past the end of array '%s[%s]'", array_str,
        free_string(array_str);
        free_string(offset_str);
}

static int known_access_ok_comparison(struct expression *expr)
{
        struct expression *array;
        struct expression *size;
        struct expression *offset;
```



```
        int comparison;

        array = get_array_base(expr);
        size = get_size_variable(array);
        if (!size)
                return 0;
        offset = get_array_offset(expr);
        comparison = get_comparison(size, offset);
        if (comparison == '>' || comparison == SPECIAL_UNSIGNED_GT)
                return 1;

        return 0;
}

static int known_access_ok_numbers(struct expression *expr)
{
        struct expression *array;
        struct expression *offset;
        sval_t max;
        int size;

        array = get_array_base(expr);
        offset = get_array_offset(expr);

        size = get_array_size(array);
        if (size <= 0)
                return 0;

        get_absolute_max(offset, &max);
        if (max.uvalue < size)
                return 1;
        return 0;
}

static void array_check_data_info(struct expression *expr)
{
        struct expression *array;
        struct expression *offset;
        struct state_list *slist;
        struct sm_state *sm;
        struct compare_data *comp;
        char *offset_name;
        const char *equal_name = NULL;

        expr = strip_expr(expr);
        if (!is_array(expr))
                return;

        if (known_access_ok_numbers(expr))
                return;
        if (known_access_ok_comparison(expr))
                return;

        array = get_array_base(expr);
        offset = get_array_offset(expr);
        offset_name = expr_to_var(offset);
        if (!offset_name)
                return;
        slist = get_all_possible_equal_comparisons(offset);
        if (!slist)
                goto free;

        FOR_EACH_PTR(slist, sm) {
                comp = sm->state->data;
                if (strcmp(comp->left_var, offset_name) == 0) {
                        if (db_var_is_array_limit(array, comp->right_var, comp->
```

```
127                                 equal_name = comp->right_var;
128                                 break;
129                         }
130                 } else if (strcmp(comp->right_var, offset_name) == 0) {
131                         if (db_var_is_array_limit(array, comp->left_var, comp->l
132                                 equal_name = comp->left_var;
133                                 break;
134                         }
135                 }
136         } END_FOR_EACH_PTR(sm);

138         if (equal_name) {
139                 char *array_name = expr_to_str(array);

141                 sm_warning("potential off by one '%s[]' limit '%s'", array_name,
142                 free_string(array_name);
143         }

145 free:
146         free_slist(&slist);
147         free_string(offset_name);
148 }

150 void check_off_by_one_relative(int id)
151 {
152         my_id = id;

154         add_hook(&array_check, OP_HOOK);
155         add_hook(&array_check_data_info, OP_HOOK);
156 }
```

```
**********************************************************
    4728 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_or_vs_and.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_function_hashtable.h"

static int my_id;

DEFINE_STRING_HASHTABLE_STATIC(unconstant_macros);

static int does_inc_dec(struct expression *expr)
{
        if (expr->type == EXPR_PREOP || expr->type == EXPR_POSTOP) {
                if (expr->op == SPECIAL_INCREMENT || expr->op == SPECIAL_DECREME
                        return 1;
                return does_inc_dec(expr->unop);
        }
        return 0;
}

static int expr_equiv_no_inc_dec(struct expression *one, struct expression *two)
{
        if (does_inc_dec(one) || does_inc_dec(two))
                return 0;
        return expr_equiv(one, two);
}

static int inconsistent_check(struct expression *left, struct expression *right)
{
        sval_t sval;

        if (get_value(left->left, &sval)) {
                if (get_value(right->left, &sval))
                        return expr_equiv_no_inc_dec(left->right, right->right);
                if (get_value(right->right, &sval))
                        return expr_equiv_no_inc_dec(left->right, right->left);
                return 0;
        }
        if (get_value(left->right, &sval)) {
                if (get_value(right->left, &sval))
                        return expr_equiv_no_inc_dec(left->left, right->right);
                if (get_value(right->right, &sval))
                        return expr_equiv_no_inc_dec(left->left, right->left);
                return 0;
        }
```



```
        return 0;
}

static void check_or(struct expression *expr)
{
        struct expression *left, *right;

        left = strip_expr(expr->left);
        right = strip_expr(expr->right);

        if (left->type != EXPR_COMPARE || left->op != SPECIAL_NOTEQUAL)
                return;
        if (right->type != EXPR_COMPARE || right->op != SPECIAL_NOTEQUAL)
                return;
        if (!inconsistent_check(left, right))
                return;

        sm_warning("was && intended here instead of ||?");
}

static int is_kernel_min_macro(struct expression *expr)
{
        char *macro;

        if (option_project != PROJ_KERNEL)
                return 0;
        macro = get_macro_name(expr->pos);
        if (!macro)
                return 0;
        if (strcmp(macro, "min") == 0 ||
            strcmp(macro, "min_t") == 0 ||
            strcmp(macro, "max") == 0 ||
            strcmp(macro, "max_t") == 0)
                return 1;
        return 0;
}

static void check_and(struct expression *expr)
{
        struct expression *left, *right;

        if (is_kernel_min_macro(expr))
                return;

        left = strip_expr(expr->left);
        right = strip_expr(expr->right);

        if (left->type != EXPR_COMPARE || left->op != SPECIAL_EQUAL)
                return;
        if (right->type != EXPR_COMPARE || right->op != SPECIAL_EQUAL)
                return;
        if (!inconsistent_check(left, right))
                return;

        sm_warning("was || intended here instead of &&?");
}

static void match_logic(struct expression *expr)
{
        if (expr->type != EXPR_LOGICAL)
                return;

        if (expr->op == SPECIAL_LOGICAL_OR)
                check_or(expr);
        if (expr->op == SPECIAL_LOGICAL_AND)
                check_and(expr);
```

```
127 }

129 static int is_unconstant_macro(struct expression *expr)
130 {
131         char *macro;

133         macro = get_macro_name(expr->pos);
134         if (!macro)
135                 return 0;
136         if (search_unconstant_macros(unconstant_macros, macro))
137                 return 1;
138         return 0;
139 }

141 static void match_condition(struct expression *expr)
142 {
143         sval_t sval;

145         if (expr->type != EXPR_BINOP)
146                 return;
147         if (expr->op == '|') {
148                 if (get_value(expr->left, &sval) || get_value(expr->right, &sval
149                         sm_warning("suspicious bitop condition");
150                 return;
151         }

153         if (expr->op != '&')
154                 return;

156         if (get_macro_name(expr->pos))
157                 return;
158         if (is_unconstant_macro(expr->left) || is_unconstant_macro(expr->right))
159                 return;

161         if ((get_value(expr->left, &sval) && sval.value == 0) ||
162             (get_value(expr->right, &sval) && sval.value == 0))
163                 sm_warning("bitwise AND condition is false here");
164 }

166 static void match_binop(struct expression *expr)
167 {
168         sval_t left, right, sval;

170         if (expr->op != '&')
171                 return;
172         if (!get_value(expr, &sval) || sval.value != 0)
173                 return;
174         if (get_macro_name(expr->pos))
175                 return;
176         if (!get_value(expr->left, &left) || !get_value(expr->right, &right))
177                 return;
178         sm_warning("odd binop '0x%llx & 0x%llx'", left.uvalue, right.uvalue);
179 }

181 void check_or_vs_and(int id)
182 {
183         my_id = id;

185         unconstant_macros = create_function_hashtable(100);
186         load_strings("unconstant_macros", unconstant_macros);

188         add_hook(&match_logic, LOGIC_HOOK);
189         add_hook(&match_condition, CONDITION_HOOK);
190         if (option_spammy)
191                 add_hook(&match_binop, BINOP_HOOK);
192 }
```

```
**********************************************************
    2535 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_param_mapper.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The idea behind this test is that if we have:
 * void foo(int bar)
 * {
 *         baz(1, bar);
 * }
 *
 * Passing "bar" to foo() really means passing "bar" to baz();
 *
 * In this case it would print:
 * info: param_mapper 0 => bar 1
 *
 */

#include "smatch.h"

static int my_id;

STATE(argument);

static struct symbol *func_sym;

static void delete(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &undefined);
}

static void match_function_def(struct symbol *sym)
{
        struct symbol *arg;

        func_sym = sym;
        FOR_EACH_PTR(func_sym->ctype.base_type->arguments, arg) {
                if (!arg->ident) {
                        continue;
                }
                set_state(my_id, arg->ident->name, arg, &argument);
        } END_FOR_EACH_PTR(arg);
}

static int get_arg_num(struct expression *expr)
{
        struct smatch_state *state;
```



```
        struct symbol *arg;
        struct symbol *this_arg;
        int i;

        expr = strip_expr(expr);
        if (expr->type != EXPR_SYMBOL)
                return -1;
        this_arg = expr->symbol;

        state = get_state_expr(my_id, expr);
        if (!state || state != &argument)
                return -1;

        i = 0;
        FOR_EACH_PTR(func_sym->ctype.base_type->arguments, arg) {
                if (arg == this_arg)
                        return i;
                i++;
        } END_FOR_EACH_PTR(arg);

        return -1;
}

static void match_call(struct expression *expr)
{
        struct expression *tmp;
        char *func;
        int arg_num;
        int i;

        if (expr->fn->type != EXPR_SYMBOL)
                return;

        func = expr->fn->symbol_name->name;

        i = 0;
        FOR_EACH_PTR(expr->args, tmp) {
                tmp = strip_expr(tmp);
                arg_num = get_arg_num(tmp);
                if (arg_num >= 0)
                        sm_msg("info: param_mapper %d => %s %d", arg_num, func,
                i++;
        } END_FOR_EACH_PTR(tmp);
}

void check_param_mapper(int id)
{
        if (!option_param_mapper)
                return;
        my_id = id;
        add_modification_hook(my_id, &delete);
        add_hook(&match_function_def, FUNC_DEF_HOOK);
        add_hook(&match_call, FUNCTION_CALL_HOOK);
}
```

```
**********************************************************
    2523 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_passes_sizeof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

#define NOBUF -2

static int my_id;

static struct expression *get_returned_expr(struct expression *expr)
{
        struct statement *stmt;

        stmt = last_ptr_list((struct ptr_list *)big_statement_stack);
        if (!stmt || stmt->type != STMT_EXPRESSION || !stmt->expression)
                return NULL;
        if (stmt->expression->type != EXPR_ASSIGNMENT)
                return NULL;
        if (stmt->expression->right != expr)
                return NULL;
        return stmt->expression->left;
}

static struct expression *remove_dereference(struct expression *expr)
{
        if (!expr || expr->type != EXPR_PREOP || expr->op != '*')
                return expr;
        expr = expr->unop;
        if (!expr || expr->type != EXPR_PREOP || expr->op != '*')
                return expr;
        return expr->unop;
}

static int get_buf_number(struct expression *call, struct expression *size_arg)
{
        struct expression *arg;
        int idx = -1;

        size_arg = strip_expr(size_arg->cast_expression);
        size_arg = remove_dereference(size_arg);

        arg = get_returned_expr(call);
        if (arg && expr_equiv(arg, size_arg))
                return idx;
```



```
        FOR_EACH_PTR(call->args, arg) {
                idx++;
                if (expr_equiv(arg, size_arg))
                        return idx;
        } END_FOR_EACH_PTR(arg);

        return NOBUF;
}

static void match_call(struct expression *call)
{
        struct expression *arg;
        char *name;
        int buf_nr;
        int i = -1;

        if (call->fn->type != EXPR_SYMBOL)
                return;

        name = expr_to_var(call->fn);
        FOR_EACH_PTR(call->args, arg) {
                i++;
                if (arg->type != EXPR_SIZEOF)
                        continue;
                buf_nr = get_buf_number(call, arg);
                if (buf_nr == NOBUF)
                        sm_msg("info: sizeof_param '%s' %d", name, i);
                else
                        sm_msg("info: sizeof_param '%s' %d %d", name, i, buf_nr);
        } END_FOR_EACH_PTR(arg);
        free_string(name);
}

void check_passes_sizeof(int id)
{
        if (!option_info)
                return;

        my_id = id;
        add_hook(&match_call, FUNCTION_CALL_HOOK);
}
```

```
**********************************************************
    2101 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_platform_device_put.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

#define MAX_ERRNO 4095

STATE(added);
STATE(not_added);

static void match_added(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *unused)
{
        struct expression *arg_expr;

        arg_expr = get_argument_from_call_expr(call_expr->args, 0);
        set_state_expr(my_id, arg_expr, &added);
}

static void match_not_added(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *unused)
{
        struct expression *arg_expr;

        arg_expr = get_argument_from_call_expr(call_expr->args, 0);
        set_state_expr(my_id, arg_expr, &not_added);
}

static void match_platform_device_del(const char *fn, struct expression *expr, v
{
        struct expression *arg_expr;
        struct sm_state *sm;

        arg_expr = get_argument_from_call_expr(expr->args, 0);
        sm = get_sm_state_expr(my_id, arg_expr);
        if (!sm)
                return;
        if (!slist_has_state(sm->possible, &not_added))
                return;
        sm_warning("perhaps platform_device_put() was intended here?");
}
```



```
void check_platform_device_put(int id)
{
        if (option_project != PROJ_KERNEL)
                return;
        my_id = id;

        return_implies_state("platform_device_add", 0, 0, &match_added, NULL);
        return_implies_state("platform_device_add", -MAX_ERRNO, -1, &match_not_a
        add_function_hook("platform_device_del", &match_platform_device_del, NUL
}
```

```
**********************************************************
    3273 Fri Dec 21 15:00:06 2018
new/usr/src/tools/smatch/src/check_pointer_math.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

STATE(size_in_bytes);

static void set_undefined(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state == &size_in_bytes)
                set_state(my_id, sm->name, sm->sym, &undefined);
}

static int is_sizeof(struct expression *expr)
{
        return (expr->type == EXPR_SIZEOF);
}

static int is_macro(struct expression *expr, const char *macro_name)
{
        char *name;
        struct expression *outside_expr;

        /* check that we aren't inside the macro itself */
        outside_expr = last_ptr_list((struct ptr_list *)big_expression_stack);
        if (outside_expr && positions_eq(expr->pos, outside_expr->pos))
                return 0;

        name = get_macro_name(expr->pos);
        if (name && strcmp(name, macro_name) == 0)
                return 1;
        return 0;
}

static int is_size_in_bytes(struct expression *expr)
{
        if (is_sizeof(expr))
                return 1;

        if (is_macro(expr, "offsetof"))
                return 1;
        if (is_macro(expr, "PAGE_SIZE"))
                return 1;
```



```
        if (get_state_expr(my_id, expr) == &size_in_bytes)
                return 1;

        return 0;
}

static void match_binop(struct expression *expr)
{
        struct symbol *type;
        char *name;
        int size;

        if (expr->op != '+')
                return;
        type = get_pointer_type(expr->left);
        if (!type)
                return;
        if (type_bits(type) <= 8) /* ignore void, bool and char pointers*/
                return;
        if (!is_size_in_bytes(expr->right))
                return;

        /* if we know it's within bounds then don't complain */
        size = get_array_size(expr->left);
        if (size) {
                sval_t max;

                get_absolute_max(expr->right, &max);
                if (max.uvalue < size)
                        return;
        }

        name = expr_to_str(expr->left);
        sm_warning("potential pointer math issue ('%s' is a %d bit pointer)",
               name, type_bits(type));
        free_string(name);
}

static void match_assign(struct expression *expr)
{
        if (expr->op != '=')
                return;

        if (!is_size_in_bytes(expr->right))
                return;
        set_state_expr(my_id, expr->left, &size_in_bytes);
}

static void check_assign(struct expression *expr)
{
        struct symbol *type;
        char *name;

        if (expr->op != SPECIAL_ADD_ASSIGN && expr->op != SPECIAL_SUB_ASSIGN)
                return;

        type = get_pointer_type(expr->left);
        if (!type)
                return;
        if (type_bits(type) == 8 || type_bits(type) == -1)
                return;
        if (!is_size_in_bytes(expr->right))
                return;
        name = expr_to_var(expr->left);
        sm_warning("potential pointer math issue ('%s' is a %d bit pointer)",
               name, type_bits(type));
```

```
127         free_string(name);
128 }

130 void check_pointer_math(int id)
131 {
132         my_id = id;
133         add_hook(&match_binop, BINOP_HOOK);
134         add_hook(&match_assign, ASSIGNMENT_HOOK);
135         add_hook(&check_assign, ASSIGNMENT_HOOK);
136         add_modification_hook(my_id, &set_undefined);
137 }
```

```
**********************************************************
    3282 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_precedence.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;

static int is_bool(struct expression *expr)
{
        struct symbol *type;

        type = get_type(expr);
        if (!type)
                return 0;
        if (type_bits(type) == 1 && type->ctype.modifiers & MOD_UNSIGNED)
                return 1;
        return 0;
}

static int is_bool_from_context(struct expression *expr)
{
        sval_t sval;

        if (!get_implied_max(expr, &sval) || sval.uvalue > 1)
                return 0;
        if (!get_implied_min(expr, &sval) || sval.value < 0)
                return 0;
        return 1;
}

static int is_bool_op(struct expression *expr)
{
        expr = strip_expr(expr);

        if (expr->type == EXPR_PREOP && expr->op == '!')
                return 1;
        if (expr->type == EXPR_COMPARE)
                return 1;
        if (expr->type == EXPR_LOGICAL)
                return 1;
        return is_bool(expr);
}

static void match_condition(struct expression *expr)
{
```



```
        int print = 0;

        if (expr->type == EXPR_COMPARE) {
                if (expr->left->type == EXPR_COMPARE || expr->right->type == EXP
                        print = 1;
                if (expr->left->type == EXPR_PREOP && expr->left->op == '!') {
                        if (expr->left->unop->type == EXPR_PREOP && expr->left->
                                return;
                        if (expr->right->op == '!')
                                return;
                        if (is_bool(expr->right))
                                return;
                        if (is_bool(expr->left->unop))
                                return;
                        if (is_bool_from_context(expr->left->unop))
                                return;
                        print = 1;
                }
        }

        if (expr->type == EXPR_BINOP) {
                if (expr->left->type == EXPR_COMPARE || expr->right->type == EXP
                        print = 1;
        }

        if (print) {
                sm_warning("add some parenthesis here?");
                return;
        }

        if (expr->type == EXPR_BINOP && expr->op == '&') {
                int i = 0;

                if (is_bool_op(expr->left))
                        i++;
                if (is_bool_op(expr->right))
                        i++;
                if (i == 1)
                        sm_warning("maybe use && instead of &");
        }
}

static void match_binop(struct expression *expr)
{
        if (expr->op != '&')
                return;
        if (expr->left->op == '!')
                sm_warning("add some parenthesis here?");
}

static void match_mask(struct expression *expr)
{
        if (expr->op != '&')
                return;
        if (expr->right->type != EXPR_BINOP)
                return;
        if (expr->right->op != SPECIAL_RIGHTSHIFT)
                return;

        sm_warning("shift has higher precedence than mask");
}

static void match_subtract_shift(struct expression *expr)
{
        if (expr->op != SPECIAL_LEFTSHIFT)
                return;
```

```
127         if (expr->right->type != EXPR_BINOP)
128                 return;
129         if (expr->right->op != '-')
130                 return;
131         sm_warning("subtract is higher precedence than shift");
132 }

134 void check_precedence(int id)
135 {
136         my_id = id;

138         add_hook(&match_condition, CONDITION_HOOK);
139         add_hook(&match_binop, BINOP_HOOK);
140         add_hook(&match_mask, BINOP_HOOK);
141         add_hook(&match_subtract_shift, BINOP_HOOK);
142 }
```

```
**********************************************************
    1814 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_proc_create.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static struct {
        int name_param;
        int mode_param;
} param_index[] = {
        {.name_param = 0, .mode_param = 1},
        {.name_param = 1, .mode_param = 2},
};

#define S_IWOTH 00002

static void match_create(const char *fn, struct expression *expr, void *_param_t
{
        struct expression *arg_expr;
        sval_t sval;
        char *name;
        int idx = PTR_INT(_param_type);

        arg_expr = get_argument_from_call_expr(expr->args, param_index[idx].mode
        if (!get_implied_value(arg_expr, &sval))
                return;
        if (!(sval.uvalue & S_IWOTH))
                return;
        arg_expr = get_argument_from_call_expr(expr->args, param_index[idx].name
        name = expr_to_var(arg_expr);
        sm_warning("proc file '%s' is world writable", name);
        free_string(name);
}

void check_proc_create(int id)
{
        my_id = id;
        if (option_project != PROJ_KERNEL)
                return;

        add_function_hook("proc_create", &match_create, INT_PTR(0));
        add_function_hook("create_proc_entry", &match_create, INT_PTR(0));
        add_function_hook("proc_create_data", &match_create, INT_PTR(0));
        add_function_hook("proc_net_fops_create", match_create, INT_PTR(1));
}
```

```
**********************************************************
    3228 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_puts_argument.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This script is for finding functions like hcd_buffer_free() which free
 * their arguments.  After running it, add those functions to check_memory.c
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(putted);

static struct symbol *this_func;
static struct tracker_list *putted_args = NULL;

static void match_function_def(struct symbol *sym)
{
        this_func = sym;
}

static int parent_is_arg(struct symbol *sym)
{
        struct symbol *arg;

        FOR_EACH_PTR(this_func->ctype.base_type->arguments, arg) {
                if (sym == arg)
                        return 1;
        } END_FOR_EACH_PTR(arg);
        return 0;
}

static void match_put(const char *fn, struct expression *expr, void *info)
{
        struct expression *tmp;
        struct symbol *sym;
        char *name;

        tmp = get_argument_from_call_expr(expr->args, 0);
        tmp = strip_expr(tmp);
        name = expr_to_var_sym(tmp, &sym);
        free_string(name);
        if (parent_is_arg(sym) && sym->ident)
                set_state(my_id, sym->ident->name, sym, &putted);
```



```
}

static int return_count = 0;
static void match_return(struct expression *ret_value)
{
        struct stree *stree;
        struct sm_state *tmp;
        struct tracker *tracker;

        if (__inline_fn)
                return;

        if (!return_count) {
                stree = __get_cur_stree();
                FOR_EACH_MY_SM(my_id, stree, tmp) {
                        if (tmp->state == &putted)
                                add_tracker(&putted_args, my_id, tmp->name,
                                            tmp->sym);
                } END_FOR_EACH_SM(tmp);
        } else {
                FOR_EACH_PTR(putted_args, tracker) {
                        tmp = get_sm_state(my_id, tracker->name, tracker->sym);
                        if (tmp && tmp->state != &putted)
                                del_tracker(&putted_args, my_id, tracker->name,
                                            tracker->sym);
                } END_FOR_EACH_PTR(tracker);

        }
}

static void print_arg(struct symbol *sym)
{
        struct symbol *arg;
        int i = 0;

        FOR_EACH_PTR(this_func->ctype.base_type->arguments, arg) {
                if (sym == arg) {
                        sm_info("puts_arg %s %d", get_function(), i);
                        return;
                }
                i++;
        } END_FOR_EACH_PTR(arg);
}

static void match_end_func(struct symbol *sym)
{
        struct tracker *tracker;

        if (__inline_fn)
                return;
        if (is_reachable())
                match_return(NULL);

        FOR_EACH_PTR(putted_args, tracker) {
                print_arg(tracker->sym);
        } END_FOR_EACH_PTR(tracker);

        free_trackers_and_list(&putted_args);
        return_count = 0;
}

void check_puts_argument(int id)
{
        if (!option_info || option_project != PROJ_KERNEL)
                return;
```

```
127         my_id = id;
128         add_hook(&match_function_def, FUNC_DEF_HOOK);
129         add_function_hook("kobject_put", &match_put, NULL);
130         add_function_hook("kref_put", &match_put, NULL);
131         add_hook(&match_return, RETURN_HOOK);
132         add_hook(&match_end_func, END_FUNC_HOOK);
133 }
```

```
**********************************************************
    3683 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_readl_infinite_loops.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;

STATE(readl);
STATE(readl_ff);
STATE(readl_00);

DECLARE_PTR_LIST(state_stack, struct smatch_state);
struct state_stack *state_at_start;

static int readl_has_been_called;
static int returned;

static int is_readl_call(struct expression *expr)
{
        struct symbol *sym;

        expr = strip_expr(expr);
        if (expr->type != EXPR_CALL)
                return 0;
        if (expr->fn->type != EXPR_SYMBOL)
                return 0;
        sym = expr->fn->symbol;
        if (!sym || !sym->ident)
                return 0;
        if (strcmp(sym->ident->name, "readl") != 0)
                return 0;
        return 1;
}

static int is_readl(struct expression *expr)
{
        if (is_readl_call(expr))
                return 1;
        if (get_state_expr(my_id, expr) == &readl)
                return 1;
        return 0;
}

static void match_assign(struct expression *expr)
{
```



```
        if (is_readl(expr->right))
                set_state_expr(my_id, expr->left, &readl);
        else if (get_state_expr(my_id, expr->left))
                set_state_expr(my_id, expr->left, &undefined);
}

static int condition_depends_on_readl(struct expression *expr)
{
        if (expr->type == EXPR_BINOP) {
                if (condition_depends_on_readl(expr->left))
                        return 1;
                if (condition_depends_on_readl(expr->right))
                        return 1;
                return 0;
        }
        if (is_readl(expr))
                return 1;
        return 0;
}

static void check_condition(struct expression *expr)
{
        if (expr->op != '&')
                return;
        if (!condition_depends_on_readl(expr))
                return;
        readl_has_been_called = 1;
        set_true_false_states(my_id, "depends on", NULL, &readl_ff, &readl_00);
}

static void match_return(struct expression *expr)
{

        if (__inline_fn)
                return;
        returned = 1;
#if 0
        struct smatch_state *tmp;

        if (!readl_has_been_called)
                return;

        FOR_EACH_PTR(state_at_start, tmp) {
                REPLACE_CURRENT_PTR(tmp, NULL);
        }
#endif
}

static void push_state_at_start(struct smatch_state *state)
{
        add_ptr_list(&state_at_start, state);
}

static struct smatch_state *pop_state_at_start(void)
{
        struct smatch_state *state;

        state = last_ptr_list((struct ptr_list *)state_at_start);
        delete_ptr_list_last((struct ptr_list **)&state_at_start);
        return state;
}

static void before_loop(struct statement *stmt)
{
        struct smatch_state *state;
```

```
127         if (!stmt || stmt->type != STMT_ITERATOR)
128                 return;
129         if (ptr_list_empty(state_at_start))
130                 returned = 0;
131         state = get_state(my_id, "depends on", NULL);
132         push_state_at_start(state);
133 }

135 static void after_loop(struct statement *stmt)
136 {
137         struct smatch_state *old_state;

139         if (!stmt || stmt->type != STMT_ITERATOR)
140                 return;
141         old_state = pop_state_at_start();
142         if (old_state == &readl_00)
143                 return;
144         if (returned)
145                 return;
146         if (get_state(my_id, "depends on", NULL) != &readl_00)
147                 return;
148         sm_warning("this loop depends on readl() succeeding");
149 }

151 void check_readl_infinite_loops(int id)
152 {
153         if (option_project != PROJ_KERNEL)
154                 return;

156         my_id = id;

158         add_hook(match_assign, ASSIGNMENT_HOOK);
159         add_hook(check_condition, CONDITION_HOOK);

161         add_hook(&match_return, RETURN_HOOK);

163         add_hook(before_loop, STMT_HOOK);
164         add_hook(after_loop, STMT_HOOK_AFTER);
165 }
```

```
**********************************************************
    2423 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_release_resource.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * I found a bug where someone released the wrong resource and wanted to
 * prevent that from happening again.
 *
 */

#include "smatch.h"

static int my_id;

static struct tracker_list *resource_list;

static void match_request(const char *fn, struct expression *expr, void *_arg_no
{
        struct expression *arg_expr;
        int arg_no = PTR_INT(_arg_no);
        char *name;
        struct symbol *sym;

        arg_expr = get_argument_from_call_expr(expr->args, arg_no);
        arg_expr = strip_expr(arg_expr);

        name = expr_to_var_sym(arg_expr, &sym);
        if (!name || !sym)
                goto free;
        add_tracker(&resource_list, my_id, name, sym);
free:
        free_string(name);
}

static void match_release(const char *fn, struct expression *expr, void *_arg_no
{
        struct expression *arg_expr;
        int arg_no = PTR_INT(_arg_no);
        char *name;
        struct symbol *sym;

        arg_expr = get_argument_from_call_expr(expr->args, arg_no);
        arg_expr = strip_expr(arg_expr);

        if (!resource_list)
                return;
```



```
        name = expr_to_var_sym(arg_expr, &sym);
        if (!name || !sym)
                goto free;
        if (in_tracker_list(resource_list, my_id, name, sym))
                goto free;
        sm_warning("'%s' was not one of the resources you requested", name);
free:
        free_string(name);
}

static void match_end_func(struct symbol *sym)
{
        if (__inline_fn)
                return;
        free_trackers_and_list(&resource_list);
}

void check_release_resource(int id)
{
        my_id = id;

        if (option_project != PROJ_KERNEL)
                return;

        add_function_hook("request_resource", &match_request, (void *)1);
        add_function_hook("release_resource", &match_release, (void *)0);
        add_function_hook("request_mem_resource", &match_request, (void *)0);
        add_function_hook("release_mem_resource", &match_release, (void *)0);
        add_hook(&match_end_func, END_FUNC_HOOK);
}
```

```
**********************************************************
    2372 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_resource_size.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;
extern int check_assigned_expr_id;

static int is_probably_ok(struct expression *expr)
{
        expr = strip_expr(expr);

        if (expr->type == EXPR_BINOP)
                return 1;
        if (expr->type == EXPR_SIZEOF)
                return 1;

        return 0;
}

static void verify_size_expr(struct expression *expr)
{
        if (expr->type != EXPR_BINOP)
                return;
        if (expr->op != '-')
                return;
        if (is_probably_ok(expr->left))
                return;
        if (is_probably_ok(expr->right))
                return;
        sm_warning("consider using resource_size() here");
}

static void handle_assigned_expr(struct expression *expr)
{
        struct smatch_state *state;

        state = get_state_expr(check_assigned_expr_id, expr);
        if (!state || !state->data)
                return;
        expr = (struct expression *)state->data;
        verify_size_expr(expr);
}

static void match_resource(const char *fn, struct expression *expr, void *_arg_n
{
```



```
        struct expression *arg_expr;
        int arg_no = PTR_INT(_arg_no);

        arg_expr = get_argument_from_call_expr(expr->args, arg_no);
        arg_expr = strip_expr(arg_expr);
        if (!arg_expr)
                return;

        if (arg_expr->type == EXPR_SYMBOL) {
                handle_assigned_expr(arg_expr);
                return;
        }
        verify_size_expr(arg_expr);
}

void check_resource_size(int id)
{
        my_id = id;

        if (option_project != PROJ_KERNEL)
                return;

        add_function_hook("ioremap_nocache", &match_resource, (void *)1);
        add_function_hook("ioremap", &match_resource, (void *)1);
        add_function_hook("__request_region", &match_resource, (void *)2);
        add_function_hook("__release_region", &match_resource, (void *)2);
        add_function_hook("__devm_request_region", &match_resource, (void *)3);
        add_function_hook("__devm_release_region", &match_resource, (void *)3);
}
```

```
**********************************************************
    1666 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_return.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static void must_check(const char *fn, struct expression *expr, void *data)
{
        struct statement *stmt;

        stmt = last_ptr_list((struct ptr_list *)big_statement_stack);
        if (stmt->type == STMT_EXPRESSION && stmt->expression == expr)
                sm_warning("unchecked '%s'", fn);
}

static void register_must_check_funcs(void)
{
        struct token *token;
        const char *func;
        static char name[256];


        snprintf(name, 256, "%s.must_check_funcs", option_project_str);
        name[255] = '\0';
        token = get_tokens_file(name);
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                func = show_ident(token->ident);
                add_function_hook(func, &must_check, NULL);
                token = token->next;
        }
        clear_token_alloc();
}

void check_return(int id)
{
        my_id = id;
        register_must_check_funcs();
```



```
}
```

**********
*1318 Fri Dec 21 15:00:07 2018*
*new/usr/src/tools/smatch/src/check_return_cast.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Complains about places that return -1 instead of -ENOMEM
 */

#include "smatch.h"

static int my_id;

static void match_return(struct expression *ret_value)
{
        struct symbol *func_type = get_real_base_type(cur_func_sym);
        sval_t sval;

        if (!func_type)
                return;
        if (!type_unsigned(func_type))
                return;
        if (type_bits(func_type) > 16)
                return;
        if (!get_fuzzy_min(ret_value, &sval))
                return;
        if (sval_is_positive(sval) || sval_cmp_val(sval, -1) == 0)
                return;

        sm_warning("signedness bug returning '%s'", sval_to_str(sval));
}

void check_return_cast(int id)
{
        my_id = id;
        add_hook(&match_return, RETURN_HOOK);
}
```

```
**********************************************************
    3625 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_return_efault.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This tries to find places which should probably return -EFAULT
 * but return the number of bytes to copy instead.
 */

#include <string.h>
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(remaining);
STATE(ok);

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state != &ok)
                set_state(my_id, sm->name, sm->sym, &ok);
}

static void match_copy(const char *fn, struct expression *expr, void *unused)
{
        if (expr->op == SPECIAL_SUB_ASSIGN)
                return;
        set_state_expr(my_id, expr->left, &remaining);
}

static void match_condition(struct expression *expr)
{
        if (!get_state_expr(my_id, expr))
                return;
        /* If the variable is zero that's ok */
        set_true_false_states_expr(my_id, expr, NULL, &ok);
}

/*
 * This function is biased in favour of print out errors.
 * The heuristic to print is:
 *    If we have a potentially positive return from copy_to_user
 *    and there is a possibility that we return negative as well
 *    then complain.
 */
```



```
static void match_return_var(struct expression *ret_value)
{
        struct smatch_state *state;
        struct sm_state *sm;
        sval_t min;

        sm = get_sm_state_expr(my_id, ret_value);
        if (!sm)
                return;
        if (!slist_has_state(sm->possible, &remaining))
                return;
        state = get_state_expr(SMATCH_EXTRA, ret_value);
        if (!state)
                return;
        if (!get_absolute_min(ret_value, &min))
                return;
        if (min.value == 0)
                return;
        sm_warning("maybe return -EFAULT instead of the bytes remaining?");
}

static void match_return_call(struct expression *ret_value)
{
        struct expression *fn;
        struct range_list *rl;
        const char *fn_name;
        char *cur_func;

        if (!ret_value || ret_value->type != EXPR_CALL)
                return;
        cur_func = get_function();
        if (!cur_func)
                return;
        if (strstr(cur_func, "_to_user") ||
            strstr(cur_func, "_from_user"))
                return;

        fn = strip_expr(ret_value->fn);
        if (fn->type != EXPR_SYMBOL)
                return;
        fn_name = fn->symbol_name->name;
        if (strcmp(fn_name, "copy_to_user") != 0 &&
            strcmp(fn_name, "__copy_to_user") != 0 &&
            strcmp(fn_name, "copy_from_user") != 0 &&
            strcmp(fn_name, "__copy_from_user"))
                return;

        rl = db_return_vals_from_str(get_function());
        if (!rl)
                return;

        if (!sval_is_negative(rl_min(rl)))
                return;
        sm_warning("maybe return -EFAULT instead of the bytes remaining?");
}

void check_return_efault(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        my_id = id;
        add_function_assign_hook("copy_to_user", &match_copy, NULL);
        add_function_assign_hook("__copy_to_user", &match_copy, NULL);
        add_function_assign_hook("copy_from_user", &match_copy, NULL);
        add_function_assign_hook("__copy_from_user", &match_copy, NULL);
```

```
127         add_function_assign_hook("clear_user", &match_copy, NULL);
128         add_hook(&match_condition, CONDITION_HOOK);
129         add_hook(&match_return_var, RETURN_HOOK);
130         add_hook(&match_return_call, RETURN_HOOK);
131         add_modification_hook(my_id, &ok_to_use);
132 }
```

```
**********************************************************
    2139 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_return_enomem.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Complains about places that return -1 instead of -ENOMEM
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static void match_return(struct expression *ret_value)
{
        struct expression *expr;
        struct sm_state *sm;
        struct stree *stree;
        sval_t sval;

        if (!ret_value)
                return;
        if (returns_unsigned(cur_func_sym))
                return;
        if (returns_pointer(cur_func_sym))
                return;
        if (!get_value(ret_value, &sval) || sval.value != -1)
                return;
        if (get_macro_name(ret_value->pos))
                return;

        stree = __get_cur_stree();

        FOR_EACH_MY_SM(SMATCH_EXTRA, stree, sm) {
                if (!estate_get_single_value(sm->state, &sval) || sval.value !=
                        continue;
                expr = get_assigned_expr_name_sym(sm->name, sm->sym);
                if (!expr)
                        continue;
                if (expr->type != EXPR_CALL || expr->fn->type != EXPR_SYMBOL)
                        continue;
                if (!expr->fn->symbol_name)
                        continue;
                /* To be honest the correct check is:
                 * if (strstr(expr->fn->symbol_name->name, "alloc"))
                 *      complain();
```



```
                 * But it generates too many warnings and it's too depressing.
                 */
                if (strcmp(expr->fn->symbol_name->name, "kmalloc") != 0 &&
                    strcmp(expr->fn->symbol_name->name, "kzalloc") != 0)
                        continue;

                sm_warning("returning -1 instead of -ENOMEM is sloppy");
                return;

        } END_FOR_EACH_SM(sm);
}

void check_return_enomem(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        my_id = id;
        add_hook(&match_return, RETURN_HOOK);
}
```

```
**********************************************************
    1455 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_return_negative_var.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static void match_return(struct expression *ret_value)
{
        struct expression *expr;
        char *macro;

        if (!ret_value)
                return;
        expr = ret_value;
        if (ret_value->type != EXPR_PREOP || ret_value->op != '-')
                return;

        macro = get_macro_name(expr->unop->pos);
        if (macro && !strcmp(macro, "PTR_ERR")) {
                sm_warning("returning -%s()", macro);
                return;
        }

        if (!option_spammy)
                return;

        expr = get_assigned_expr(ret_value->unop);
        if (!expr)
                return;
        if (expr->type != EXPR_CALL)
                return;

        sm_warning("should this return really be negated?");
}

void check_return_negative_var(int id)
{
        if (option_project != PROJ_KERNEL)
                return;

        my_id = id;
        add_hook(&match_return, RETURN_HOOK);
}
```

```
**********************************************************
    8998 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_rosenberg.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/* Does a search for Dan Rosenberg style info leaks */

/* fixme: struct includes a struct with a hole in it */
/* function is called that clears the struct */

#include "scope.h"
#include "smatch.h"
#include "smatch_function_hashtable.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_whole_id;
static int my_member_id;

STATE(cleared);

static void extra_mod_hook(const char *name, struct symbol *sym, struct expressi
{
        struct symbol *type;

        type = get_real_base_type(sym);
        if (!type || type->type != SYM_STRUCT)
                return;

        set_state(my_member_id, name, sym, state);
}

static void print_holey_warning(struct expression *data, const char *member)
{
        char *name;

        name = expr_to_str(data);
        if (member) {
                sm_warning("check that '%s' doesn't leak information (struct has
                        name, member);
        } else {
                sm_warning("check that '%s' doesn't leak information (struct has
                        name);
        }
        free_string(name);
}

static int check_struct(struct expression *expr, struct symbol *type)
```



```
{
        struct symbol *tmp, *base_type;
        const char *prev = NULL;
        int align;

        if (type->ctype.alignment == 1)
                return 0;

        align = 0;
        FOR_EACH_PTR(type->symbol_list, tmp) {
                base_type = get_real_base_type(tmp);
                if (base_type && base_type->type == SYM_STRUCT) {
                        if (check_struct(expr, base_type))
                                return 1;
                }

                if (!tmp->ctype.alignment) {
                        sm_perror("cannot determine the alignment here");
                } else if (align % tmp->ctype.alignment) {
                        print_holey_warning(expr, prev);
                        return 1;
                }

                if (base_type == &bool_ctype)
                        align += 1;
                else if (type_bits(tmp) <= 0)
                        align = 0;
                else
                        align += type_bytes(tmp);

                if (tmp->ident)
                        prev = tmp->ident->name;
                else
                        prev = NULL;
        } END_FOR_EACH_PTR(tmp);

        if (align % type->ctype.alignment) {
                print_holey_warning(expr, prev);
                return 1;
        }

        return 0;
}

static int warn_on_holey_struct(struct expression *expr)
{
        struct symbol *type;
        type = get_type(expr);
        if (!type || type->type != SYM_STRUCT)
                return 0;

        return check_struct(expr, type);
}

static int has_global_scope(struct expression *expr)
{
        struct symbol *sym;

        if (expr->type != EXPR_SYMBOL)
                return FALSE;
        sym = expr->symbol;
        if (!sym)
                return FALSE;
        return toplevel(sym->scope);
}
```

```
127 static int was_initialized(struct expression *expr)
128 {
129         struct symbol *sym;
130         char *name;
131
132         name = expr_to_var_sym(expr, &sym);
133         if (!name)
134                 return 0;
135         if (sym->initializer)
136                 return 1;
137         return 0;
138 }
139
140 static void match_clear(const char *fn, struct expression *expr, void *_arg_no)
141 {
142         struct expression *ptr;
143         int arg_no = PTR_INT(_arg_no);
144
145         ptr = get_argument_from_call_expr(expr->args, arg_no);
146         if (!ptr)
147                 return;
148         ptr = strip_expr(ptr);
149         if (ptr->type != EXPR_PREOP || ptr->op != '&')
150                 return;
151         ptr = strip_expr(ptr->unop);
152         set_state_expr(my_whole_id, ptr, &cleared);
153 }
154
155 static int was_memset(struct expression *expr)
156 {
157         if (get_state_expr(my_whole_id, expr) == &cleared)
158                 return 1;
159         return 0;
160 }
161
162 static int member_initialized(char *name, struct symbol *outer, struct symbol *m
163 {
164         char buf[256];
165         struct symbol *base;
166
167         base = get_base_type(member);
168         if (!base || base->type != SYM_BASETYPE || !member->ident)
169                 return FALSE;
170
171         if (pointer)
172                 snprintf(buf, 256, "%s->%s", name, member->ident->name);
173         else
174                 snprintf(buf, 256, "%s.%s", name, member->ident->name);
175
176         if (get_state(my_member_id, buf, outer))
177                 return TRUE;
178
179         return FALSE;
180 }
181
182 static int member_uninitialized(char *name, struct symbol *outer, struct symbol
183 {
184         char buf[256];
185         struct symbol *base;
186         struct sm_state *sm;
187
188         base = get_base_type(member);
189         if (!base || base->type != SYM_BASETYPE || !member->ident)
190                 return FALSE;
191
192         if (pointer)
```



```
193                 snprintf(buf, 256, "%s->%s", name, member->ident->name);
194         else
195                 snprintf(buf, 256, "%s.%s", name, member->ident->name);
196
197         sm = get_sm_state(my_member_id, buf, outer);
198         if (sm && !slist_has_state(sm->possible, &undefined))
199                 return FALSE;
200
201         sm_warning("check that '%s' doesn't leak information", buf);
202         return TRUE;
203 }
204
205 static int check_members_initialized(struct expression *expr)
206 {
207         char *name;
208         struct symbol *outer;
209         struct symbol *sym;
210         struct symbol *tmp;
211         int pointer = 0;
212         int printed = 0;
213
214         sym = get_type(expr);
215         if (sym && sym->type == SYM_PTR) {
216                 pointer = 1;
217                 sym = get_real_base_type(sym);
218         }
219         if (!sym)
220                 return 0;
221         if (sym->type != SYM_STRUCT)
222                 return 0;
223
224         name = expr_to_var_sym(expr, &outer);
225
226         /*
227          * check that at least one member was set.  If all of them were not set
228          * it's more likely a problem in the check than a problem in the kernel
229          * code.
230          */
231         FOR_EACH_PTR(sym->symbol_list, tmp) {
232                 if (member_initialized(name, outer, tmp, pointer))
233                         goto check;
234         } END_FOR_EACH_PTR(tmp);
235         goto out;
236
237 check:
238         FOR_EACH_PTR(sym->symbol_list, tmp) {
239                 if (member_uninitialized(name, outer, tmp, pointer)) {
240                         printed = 1;
241                         goto out;
242                 }
243         } END_FOR_EACH_PTR(tmp);
244 out:
245         free_string(name);
246         return printed;
247 }
248
249 static void check_was_initialized(struct expression *data)
250 {
251         data = strip_expr(data);
252         if (!data)
253                 return;
254         if (data->type == EXPR_PREOP && data->op == '&')
255                 data = strip_expr(data->unop);
256         if (data->type != EXPR_SYMBOL)
257                 return;
```

```
259         if (has_global_scope(data))
260                 return;
261         if (was_initialized(data))
262                 return;
263         if (was_memset(data))
264                 return;
265         if (warn_on_holey_struct(data))
266                 return;
267         check_members_initialized(data);
268 }

270 static void match_copy_to_user(const char *fn, struct expression *expr, void *_a
271 {
272         int arg = PTR_INT(_arg);
273         struct expression *data;

275         data = get_argument_from_call_expr(expr->args, arg);
276         data = strip_expr(data);
277         if (!data)
278                 return;
279         if (data->type != EXPR_PREOP || data->op != '&')
280                 return;
281         check_was_initialized(data);
282 }

284 static void db_param_cleared(struct expression *expr, int param, char *key, char
285 {
286         while (expr->type == EXPR_ASSIGNMENT)
287                 expr = strip_expr(expr->right);
288         if (expr->type != EXPR_CALL)
289                 return;

291         match_clear(NULL, expr, INT_PTR(param));
292 }

294 static void match_assign(struct expression *expr)
295 {
296         struct symbol *type;

298         type = get_type(expr->left);
299         if (!type || type->type != SYM_STRUCT)
300                 return;
301         set_state_expr(my_whole_id, expr->left, &cleared);
302 }

304 static void register_clears_argument(void)
305 {
306         struct token *token;
307         const char *func;
308         int arg;

310         token = get_tokens_file("kernel.clears_argument");
311         if (!token)
312                 return;
313         if (token_type(token) != TOKEN_STREAMBEGIN)
314                 return;
315         token = token->next;
316         while (token_type(token) != TOKEN_STREAMEND) {
317                 if (token_type(token) != TOKEN_IDENT)
318                         return;
319                 func = show_ident(token->ident);
320                 token = token->next;
321                 if (token_type(token) != TOKEN_NUMBER)
322                         return;
323                 arg = atoi(token->number);
```



```
325                 add_function_hook(func, &match_clear, INT_PTR(arg));
326                 token = token->next;
327         }
328         clear_token_alloc();
329 }

331 static void register_copy_funcs_from_file(void)
332 {
333         struct token *token;
334         const char *func;
335         int arg;

337         token = get_tokens_file("kernel.rosenberg_funcs");
338         if (!token)
339                 return;
340         if (token_type(token) != TOKEN_STREAMBEGIN)
341                 return;
342         token = token->next;
343         while (token_type(token) != TOKEN_STREAMEND) {
344                 if (token_type(token) != TOKEN_IDENT)
345                         return;
346                 func = show_ident(token->ident);
347                 token = token->next;
348                 if (token_type(token) != TOKEN_NUMBER)
349                         return;
350                 arg = atoi(token->number);
351                 add_function_hook(func, &match_copy_to_user, INT_PTR(arg));
352                 token = token->next;
353         }
354         clear_token_alloc();
355 }

357 void check_rosenberg(int id)
358 {
359         if (option_project != PROJ_KERNEL)
360                 return;
361         my_whole_id = id;

363         add_function_hook("memset", &match_clear, INT_PTR(0));
364         add_function_hook("memcpy", &match_clear, INT_PTR(0));
365         add_function_hook("memzero", &match_clear, INT_PTR(0));
366         add_function_hook("__memset", &match_clear, INT_PTR(0));
367         add_function_hook("__memcpy", &match_clear, INT_PTR(0));
368         add_function_hook("__memzero", &match_clear, INT_PTR(0));
369         add_function_hook("__builtin_memset", &match_clear, INT_PTR(0));
370         add_function_hook("__builtin_memcpy", &match_clear, INT_PTR(0));

372         add_hook(&match_assign, ASSIGNMENT_HOOK);
373         register_clears_argument();
374         select_return_states_hook(PARAM_CLEARED, &db_param_cleared);

376         register_copy_funcs_from_file();
377 }

379 void check_rosenberg2(int id)
380 {
381         if (option_project != PROJ_KERNEL)
382                 return;

384         my_member_id = id;
385         add_extra_mod_hook(&extra_mod_hook);
386 }
```

```
**********************************************************
    1063 Fri Dec 21 15:00:07 2018
new/usr/src/tools/smatch/src/check_select.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static void match_select(struct expression *expr)
{
        if (expr->cond_true)
                return;
        expr = strip_expr(expr->conditional);
        if (expr->type != EXPR_COMPARE)
                return;
        sm_warning("boolean comparison inside select");
}

void check_select(int id)
{
        my_id = id;
        add_hook(&match_select, SELECT_HOOK);
}
```

```
**********************************************************
    2167 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_shift_to_zero.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_binop(struct expression *expr)
{
        struct symbol *type;
        sval_t bits;

        if (expr->op != SPECIAL_RIGHTSHIFT)
                return;

        if (!get_implied_value(expr->right, &bits))
                return;

        type = get_type(expr->left);
        if (!type)
                return;
        if (type_bits(type) == -1 || type_bits(type) > bits.value)
                return;
        sm_warning("right shifting more than type allows %d vs %lld", type_bits(
}

static void match_binop2(struct expression *expr)
{
        struct expression *left;
        struct expression *tmp;
        sval_t mask, shift;

        if (expr->op != SPECIAL_RIGHTSHIFT)
                return;

        left = strip_expr(expr->left);
        tmp = get_assigned_expr(left);
        if (tmp)
                left = tmp;
        if (left->type != EXPR_BINOP || left->op != '&')
                return;

        if (!get_implied_value(expr->right, &shift))
                return;
        if (!get_implied_value(left->right, &mask))
                return;
```



```
        if (mask.uvalue >> shift.uvalue)
                return;

        sm_warning("mask and shift to zero");
}

static void match_assign(struct expression *expr)
{
        struct symbol *type;
        sval_t bits;

        if (expr->op != SPECIAL_SHR_ASSIGN)
                return;

        if (!get_implied_value(expr->right, &bits))
                return;
        type = get_type(expr->left);
        if (!type)
                return;
        if (type_bits(type) > bits.value)
                return;
        sm_warning("right shift assign to zero");
}

void check_shift_to_zero(int id)
{
        my_id = id;

        add_hook(&match_binop, BINOP_HOOK);
        add_hook(&match_binop2, BINOP_HOOK);

        add_hook(&match_assign, ASSIGNMENT_HOOK);

}
```

```
**********************************************************
    7585 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_signed.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Check for things which are signed but probably should be unsigned.
 *
 * Hm...  It seems like at this point in the processing, sparse makes all
 * bitfields unsigned.  Which is logical but not what GCC does.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;

#define VAR_ON_RIGHT 0
#define VAR_ON_LEFT 1

static void match_assign(struct expression *expr)
{
        struct symbol *sym;
        sval_t sval;
        sval_t max;
        sval_t min;
        char *left_name, *right_name;

        if (__in_fake_assign)
                return;
        if (expr->op == SPECIAL_AND_ASSIGN || expr->op == SPECIAL_OR_ASSIGN)
                return;

        sym = get_type(expr->left);
        if (!sym || sym->type != SYM_BASETYPE) {
                //sm_msg("could not get type");
                return;
        }
        if (type_bits(sym) < 0 || type_bits(sym) >= 32) /* max_val limits this *
                return;
        if (!get_implied_value(expr->right, &sval))
                return;
        max = sval_type_max(sym);
        if (sym != &bool_ctype && sym != &uchar_ctype &&
            sval_cmp(max, sval) < 0 &&
            !(sval.value < 256 && max.value == 127)) {
                left_name = expr_to_str(expr->left);
```



```
                right_name = expr_to_str(expr->right);
                sm_warning("'%s' %s can't fit into %s '%s'",
                        right_name, sval_to_numstr(sval), sval_to_numstr(max), le
                free_string(left_name);
        }
        min = sval_type_min(sym);
        if (sval_cmp_t(&llong_ctype, min, sval) > 0) {
                if (min.value == 0 && sval.value == -1) /* assigning -1 to unsig
                        return;
                if (expr->right->type == EXPR_PREOP && expr->right->op == '~')
                        return;
                if (expr->op == SPECIAL_SUB_ASSIGN || expr->op == SPECIAL_ADD_AS
                        return;
                if (sval_positive_bits(sval) == 7)
                        return;
                left_name = expr_to_str(expr->left);
                if (min.value == 0) {
                        sm_warning("assigning %s to unsigned variable '%s'",
                                sval_to_str(sval), left_name);
                } else {
                        sm_warning("value %s can't fit into %s '%s'",
                                sval_to_str(sval), sval_to_str(min), left_name);
                }
                free_string(left_name);
        }
}

static int cap_gt_zero_and_lt(struct expression *expr)
{

        struct expression *var = expr->left;
        struct expression *tmp;
        char *name1 = NULL;
        char *name2 = NULL;
        sval_t known;
        int ret = 0;
        int i;

        if (!get_value(expr->right, &known) || known.value != 0)
                return 0;

        i = 0;
        FOR_EACH_PTR_REVERSE(big_expression_stack, tmp) {
                if (!i++)
                        continue;
                if (tmp->op == SPECIAL_LOGICAL_AND) {
                        struct expression *right = strip_expr(tmp->right);

                        if (right->op != '<' &&
                            right->op != SPECIAL_UNSIGNED_LT &&
                            right->op != SPECIAL_LTE &&
                            right->op != SPECIAL_UNSIGNED_LTE)
                                return 0;

                        name1 = expr_to_str(var);
                        if (!name1)
                                goto free;

                        name2 = expr_to_str(right->left);
                        if (!name2)
                                goto free;
                        if (!strcmp(name1, name2))
                                ret = 1;
                        goto free;

                }
```

```
127                 return 0;
128         } END_FOR_EACH_PTR_REVERSE(tmp);

130 free:
131         free_string(name1);
132         free_string(name2);
133         return ret;
134 }

136 static int cap_lt_zero_or_gt(struct expression *expr)
137 {

139         struct expression *var = expr->left;
140         struct expression *tmp;
141         char *name1 = NULL;
142         char *name2 = NULL;
143         sval_t known;
144         int ret = 0;
145         int i;

147         if (!get_value(expr->right, &known) || known.value != 0)
148                 return 0;

150         i = 0;
151         FOR_EACH_PTR_REVERSE(big_expression_stack, tmp) {
152                 if (!i++)
153                         continue;
154                 if (tmp->op == SPECIAL_LOGICAL_OR) {
155                         struct expression *right = strip_expr(tmp->right);

157                         if (right->op != '>' &&
158                             right->op != SPECIAL_UNSIGNED_GT &&
159                             right->op != SPECIAL_GTE &&
160                             right->op != SPECIAL_UNSIGNED_GTE)
161                                 return 0;

163                         name1 = expr_to_str(var);
164                         if (!name1)
165                                 goto free;

167                         name2 = expr_to_str(right->left);
168                         if (!name2)
169                                 goto free;
170                         if (!strcmp(name1, name2))
171                                 ret = 1;
172                         goto free;

174                 }
175                 return 0;
176         } END_FOR_EACH_PTR_REVERSE(tmp);

178 free:
179         free_string(name1);
180         free_string(name2);
181         return ret;
182 }

184 static int cap_both_sides(struct expression *expr)
185 {
186         switch (expr->op) {
187         case '<':
188         case SPECIAL_UNSIGNED_LT:
189         case SPECIAL_LTE:
190         case SPECIAL_UNSIGNED_LTE:
191                 return cap_lt_zero_or_gt(expr);
192         case '>':
```



```
193         case SPECIAL_UNSIGNED_GT:
194         case SPECIAL_GTE:
195         case SPECIAL_UNSIGNED_GTE:
196                 return cap_gt_zero_and_lt(expr);
197         }
198         return 0;
199 }

201 static int compare_against_macro(struct expression *expr)
202 {
203         sval_t known;

205         if (expr->op != SPECIAL_UNSIGNED_LT)
206                 return 0;

208         if (!get_value(expr->right, &known) || known.value != 0)
209                 return 0;
210         return !!get_macro_name(expr->right->pos);
211 }

213 static int print_unsigned_never_less_than_zero(struct expression *expr)
214 {
215         sval_t known;
216         char *name;

218         if (expr->op != SPECIAL_UNSIGNED_LT)
219                 return 0;

221         if (!get_value(expr->right, &known) || known.value != 0)
222                 return 0;

224         name = expr_to_str(expr->left);
225         sm_warning("unsigned '%s' is never less than zero.", name);
226         free_string(name);
227         return 1;
228 }

230 static void match_condition(struct expression *expr)
231 {
232         struct symbol *type;
233         sval_t known;
234         sval_t min, max;
235         struct range_list *rl_left_orig, *rl_right_orig;
236         struct range_list *rl_left, *rl_right;

238         if (expr->type != EXPR_COMPARE)
239                 return;

241         type = get_type(expr);
242         if (!type)
243                 return;

245         /* screw it.  I am writing this to mark yoda code as buggy.
246          * Valid comparisons between an unsigned and zero are:
247          * 1) inside a macro.
248          * 2) foo < LOWER_BOUND where LOWER_BOUND is a macro.
249          * 3) foo < 0 || foo > X in exactly this format.  No Yoda.
250          * 4) foo >= 0 && foo < X
251          */
252         if (get_macro_name(expr->pos))
253                 return;
254         if (compare_against_macro(expr))
255                 return;
256         if (cap_both_sides(expr))
257                 return;
```

```
259         /* This is a special case for the common error */
260         if (print_unsigned_never_less_than_zero(expr))
261                 return;

263         /* check that one and only one side is known */
264         if (get_value(expr->left, &known)) {
265                 if (get_value(expr->right, &known))
266                         return;
267                 rl_left_orig = alloc_rl(known, known);
268                 rl_left = cast_rl(type, rl_left_orig);

270                 min = sval_type_min(get_type(expr->right));
271                 max = sval_type_max(get_type(expr->right));
272                 rl_right_orig = alloc_rl(min, max);
273                 rl_right = cast_rl(type, rl_right_orig);
274         } else if (get_value(expr->right, &known)) {
275                 rl_right_orig = alloc_rl(known, known);
276                 rl_right = cast_rl(type, rl_right_orig);

278                 min = sval_type_min(get_type(expr->left));
279                 max = sval_type_max(get_type(expr->left));
280                 rl_left_orig = alloc_rl(min, max);
281                 rl_left = cast_rl(type, rl_left_orig);
282         } else {
283                 return;
284         }

286         if (!possibly_true_rl(rl_left, expr->op, rl_right)) {
287                 char *name = expr_to_str(expr);

289                 sm_warning("impossible condition '(%s) => (%s %s %s)'", name,
290                         show_rl(rl_left), show_special(expr->op),
291                         show_rl(rl_right));
292                 free_string(name);
293         }

295         if (!possibly_false_rl(rl_left, expr->op, rl_right)) {
296                 char *name = expr_to_str(expr);

298                 sm_warning("always true condition '(%s) => (%s %s %s)'", name,
299                         show_rl(rl_left_orig), show_special(expr->op),
300                         show_rl(rl_right_orig));
301                 free_string(name);
302         }
303 }

305 void check_signed(int id)
306 {
307         my_id = id;

309         add_hook(&match_assign, ASSIGNMENT_HOOK);
310         add_hook(&match_condition, CONDITION_HOOK);
311 }
```

```
**********************************************************
    2522 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_signed_integer_overflow_check.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Looks for integers that we get from the user which can be attacked
 * with an integer overflow.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static void match_condition(struct expression *expr)
{
        struct expression *left, *right;
        struct symbol *type;
        char *right_name;
        char *left_name;

        if (expr->type != EXPR_COMPARE)
                return;
        if (expr->op != '<')
                return;

        type = get_type(expr);
        if (!type_signed(type))
                return;

        left = strip_expr(expr->left);
        right = strip_expr(expr->right);

        if (left->type != EXPR_BINOP) {
                left = get_assigned_expr(left);
                left = strip_expr(left);
                if (!left || left->type != EXPR_BINOP)
                        return;
        }

        if (left->op != '+' && left->op != '*' && left->op != SPECIAL_LEFTSHIFT)
                return;

        if (has_variable(left, right) == 1) {
                left_name = expr_to_str(left);
                right_name = expr_to_str(right);
```



```
                sm_warning("signed overflow undefined. '%s %s %s'", left_name, s
                free_string(left_name);
                free_string(right_name);
        }
}

static void match_binop(struct expression *expr)
{
        sval_t left_val, right_min;
        char *str;

        if (expr->op != '-')
                return;

        if (!get_value(expr->left, &left_val))
                return;

        switch (left_val.uvalue) {
        case SHRT_MAX:
        case USHRT_MAX:
        case INT_MAX:
        case UINT_MAX:
        case LLONG_MAX:
        case ULLONG_MAX:
                break;
        default:
                return;
        }

        get_absolute_min(expr->right, &right_min);
        if (!sval_is_negative(right_min))
                return;

        str = expr_to_str(expr);
        sm_warning("potential negative subtraction from max '%s'", str);
        free_string(str);
}

void check_signed_integer_overflow_check(int id)
{
        my_id = id;

        if (option_project == PROJ_KERNEL) {
                /* The kernel uses -fno-strict-overflow so it's fine */
                return;
        }

        add_hook(&match_condition, CONDITION_HOOK);
        add_hook(&match_binop, BINOP_HOOK);
}
```

```
**********************************************************
    3844 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_sizeof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void check_pointer(struct expression *expr, char *ptr_name)
{
        char *name;
        sval_t sval;

        if (!expr || expr->type != EXPR_SIZEOF)
                return;

        get_value(expr, &sval);

        expr = strip_expr(expr->cast_expression);
        name = expr_to_str(expr);
        if (!name)
                return;

        if (strcmp(ptr_name, name) == 0)
                sm_warning("was 'sizeof(*%s)' intended?", ptr_name);
        free_string(name);
}

static void match_call_assignment(struct expression *expr)
{
        struct expression *call = strip_expr(expr->right);
        struct expression *arg;
        char *ptr_name;

        if (!is_pointer(expr->left))
                return;

        ptr_name = expr_to_str(expr->left);
        if (!ptr_name)
                return;

        FOR_EACH_PTR(call->args, arg) {
                check_pointer(arg, ptr_name);
        } END_FOR_EACH_PTR(arg);

        free_string(ptr_name);
}
```



```
static void check_passes_pointer(char *name, struct expression *call)
{
        struct expression *arg;
        char *ptr_name;

        FOR_EACH_PTR(call->args, arg) {
                ptr_name = expr_to_var(arg);
                if (!ptr_name)
                        continue;
                if (strcmp(name, ptr_name) == 0)
                        sm_warning("was 'sizeof(*%s)' intended?", name);
                free_string(ptr_name);
        } END_FOR_EACH_PTR(arg);
}

static void match_check_params(struct expression *call)
{
        struct expression *arg;
        struct expression *obj;
        char *name;

        FOR_EACH_PTR(call->args, arg) {
                if (arg->type != EXPR_SIZEOF)
                        continue;
                obj = strip_expr(arg->cast_expression);
                if (!is_pointer(obj))
                        continue;
                name = expr_to_var(obj);
                if (!name)
                        continue;
                check_passes_pointer(name, call);
                free_string(name);
        } END_FOR_EACH_PTR(arg);
}

static struct string_list *macro_takes_sizeof_argument;
static void check_sizeof_number(struct expression *expr)
{
        char *macro, *tmp;

        if (expr->type != EXPR_VALUE)
                return;
        macro = get_macro_name(expr->pos);
        FOR_EACH_PTR(macro_takes_sizeof_argument, tmp) {
                if (macro && strcmp(tmp, macro) == 0)
                        return;
        } END_FOR_EACH_PTR(tmp);

        sm_warning("sizeof(NUMBER)?");
}

static void match_sizeof(struct expression *expr)
{
        check_sizeof_number(expr);
        if (expr->type == EXPR_PREOP && expr->op == '&')
                sm_warning("sizeof(&pointer)?");
        if (expr->type == EXPR_SIZEOF)
                sm_warning("sizeof(sizeof())?");
        /* the ilog2() macro is a valid place to check the size of a binop */
        if (expr->type == EXPR_BINOP && !get_macro_name(expr->pos))
                sm_warning("taking sizeof binop");
}

static void register_macro_takes_sizeof_argument(void)
{
```

```
127         struct token *token;
128         char *macro;
129         char name[256];

131         snprintf(name, 256, "%s.macro_takes_sizeof_argument", option_project_str

133         token = get_tokens_file(name);
134         if (!token)
135                 return;
136         if (token_type(token) != TOKEN_STREAMBEGIN)
137                 return;
138         token = token->next;
139         while (token_type(token) != TOKEN_STREAMEND) {
140                 if (token_type(token) != TOKEN_IDENT)
141                         return;
142                 macro = alloc_string(show_ident(token->ident));
143                 add_ptr_list(&macro_takes_sizeof_argument, macro);
144                 token = token->next;
145         }
146         clear_token_alloc();
147 }

149 void check_sizeof(int id)
150 {
151         my_id = id;

153         register_macro_takes_sizeof_argument();
154         add_hook(&match_call_assignment, CALL_ASSIGNMENT_HOOK);
155         add_hook(&match_check_params, FUNCTION_CALL_HOOK);
156         add_hook(&match_sizeof, SIZEOF_HOOK);
157 }
```

```
**********************************************************
    2264 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_snprintf.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &undefined);
}

static void match_snprintf(const char *fn, struct expression *expr, void *info)
{
        struct expression *call;
        struct expression *arg;
        sval_t buflen;

        call = strip_expr(expr->right);
        arg = get_argument_from_call_expr(call->args, 1);
        if (!get_fuzzy_max(arg, &buflen))
                return;
        set_state_expr(my_id, expr->left, alloc_state_num(buflen.value));
}

static int get_old_buflen(struct sm_state *sm)
{
        struct sm_state *tmp;
        int ret = 0;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (PTR_INT(tmp->state->data) > ret)
                        ret = PTR_INT(tmp->state->data);
        } END_FOR_EACH_PTR(tmp);
        return ret;
}

static void match_call(struct expression *expr)
{
        struct expression *arg;
        struct sm_state *sm;
        int old_buflen;
        sval_t max;
```



```
        FOR_EACH_PTR(expr->args, arg) {
                sm = get_sm_state_expr(my_id, arg);
                if (!sm)
                        continue;
                old_buflen = get_old_buflen(sm);
                if (!old_buflen)
                        return;
                if (get_absolute_max(arg, &max) && sval_cmp_val(max, old_buflen)
                        sm_warning("'%s' returned from snprintf() might be large
                                sm->name, old_buflen);
        } END_FOR_EACH_PTR(arg);
}

void check_snprintf(int id)
{
        if (option_project != PROJ_KERNEL)
                return;
        if (!option_spammy)
                return;

        my_id = id;
        add_hook(&match_call, FUNCTION_CALL_HOOK);
        add_function_assign_hook("snprintf", &match_snprintf, NULL);
        add_modification_hook(my_id, &ok_to_use);
}
```

```
**********************************************************
    2049 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_snprintf_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static void match_snprintf(const char *fn, struct expression *expr, void *unused
{
        struct expression *dest;
        struct expression *dest_size_expr;
        struct expression *format_string;
        struct expression *data;
        char *data_name = NULL;
        int dest_size;
        sval_t limit_size;
        char *format;
        int data_size;

        dest = get_argument_from_call_expr(expr->args, 0);
        dest_size_expr = get_argument_from_call_expr(expr->args, 1);
        format_string = get_argument_from_call_expr(expr->args, 2);
        data = get_argument_from_call_expr(expr->args, 3);

        dest_size = get_array_size_bytes(dest);
        if (!get_implied_value(dest_size_expr, &limit_size))
                return;
        if (dest_size > 1 && dest_size < limit_size.value)
                sm_error("snprintf() is printing too much %s vs %d",
                         sval_to_str(limit_size), dest_size);
        format = expr_to_var(format_string);
        if (!format)
                return;
        if (strcmp(format, "\"%s\""))
                goto free;
        data_name = expr_to_str(data);
        data_size = get_size_from_strlen(data);
        if (!data_size)
                data_size = get_array_size_bytes(data);
        if (limit_size.value < data_size)
                sm_error("snprintf() chops off the last chars of '%s': %d vs %s"
                       data_name, data_size, sval_to_str(limit_size));
free:
        free_string(data_name);
        free_string(format);
}

void check_snprintf_overflow(int id)
```



```
{
        add_function_hook("snprintf", &match_snprintf, NULL);
}
```

```
**********************************************************
    4550 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_spectre.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2018 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;

static int suppress_multiple = 1;

static int is_write(struct expression *expr)
{
        return 0;
}

static int is_read(struct expression *expr)
{
        struct expression *parent, *last_parent;
        struct statement *stmt;

        if (is_write(expr))
                return 0;

        last_parent = expr;
        while ((parent = expr_get_parent_expr(expr))){

                last_parent = parent;

                /* If we pass a value as a parameter that's a read, probably? */
//              if (parent->type == EXPR_CALL)
//                      return 1;

                if (parent->type == EXPR_ASSIGNMENT) {
                        if (parent->right == expr)
                                return 1;
                        if (parent->left == expr)
                                return 0;
                }
                expr = parent;
        }

        stmt = expr_get_parent_stmt(last_parent);
        if (stmt && stmt->type == STMT_RETURN)
                return 1;

        return 0;
```



```
}

static int is_harmless(struct expression *expr)
{
        struct expression *tmp, *parent;
        struct statement *stmt;
        int count = 0;

        parent = expr;
        while ((tmp = expr_get_parent_expr(parent))) {
                if (tmp->type == EXPR_ASSIGNMENT || tmp->type == EXPR_CALL)
                        return 0;
                parent = tmp;
                if (count++ > 4)
                        break;
        }

        stmt = expr_get_parent_stmt(parent);
        if (!stmt)
                return 0;
        if (stmt->type == STMT_IF && stmt->if_conditional == parent)
                return 1;
        if (stmt->type == STMT_ITERATOR &&
            (stmt->iterator_pre_condition == parent ||
             stmt->iterator_post_condition == parent))
                return 1;

        return 0;
}

static unsigned long long get_max_by_type(struct expression *expr)
{
        struct symbol *type;
        int cnt = 0;
        sval_t max;

        max.type = &ullong_ctype;
        max.uvalue = -1ULL;

        while (true) {
                expr = strip_parens(expr);
                type = get_type(expr);
                if (type && sval_type_max(type).uvalue < max.uvalue)
                        max = sval_type_max(type);
                if (expr->type == EXPR_PREOP) {
                        expr = expr->unop;
                } else if (expr->type == EXPR_BINOP) {
                        if (expr->op == '%' || expr->op == '&')
                                expr = expr->right;
                        else
                                return max.uvalue;
                } else {
                        expr = get_assigned_expr(expr);
                        if (!expr)
                                return max.uvalue;
                }
                if (cnt++ > 5)
                        return max.uvalue;
        }

        return max.uvalue;
}

static unsigned long long get_mask(struct expression *expr)
{
        struct expression *tmp;
```

```
127         sval_t mask;
128         int cnt = 0;

130         expr = strip_expr(expr);

132         tmp = get_assigned_expr(expr);
133         while (tmp) {
134                 expr = tmp;
135                 if (++cnt > 3)
136                         break;
137                 tmp = get_assigned_expr(expr);
138         }

140         if (expr->type == EXPR_BINOP && expr->op == '&') {
141                 if (get_value(expr->right, &mask))  /* right is the common case
142                         return mask.uvalue;
143                 if (get_value(expr->left, &mask))
144                         return mask.uvalue;
145         }

147         return ULLONG_MAX;
148 }

150 static void array_check(struct expression *expr)
151 {
152         struct expression_list *conditions;
153         struct expression *array_expr, *offset;
154         unsigned long long mask;
155         int array_size;
156         char *name;

158         expr = strip_expr(expr);
159         if (!is_array(expr))
160                 return;

162         if (is_impossible_path())
163                 return;
164         if (is_harmless(expr))
165                 return;

167         array_expr = get_array_base(expr);
168         if (suppress_multiple && is_ignored_expr(my_id, array_expr))
169                 return;

171         offset = get_array_offset(expr);
172         if (!is_user_rl(offset))
173                 return;
174         if (is_nospec(offset))
175                 return;

177         array_size = get_array_size(array_expr);
178         if (array_size > 0 && get_max_by_type(offset) < array_size)
179                 return;
180 //      binfo = get_bit_info(offset);
181 //      if (array_size > 0 && binfo && binfo->possible < array_size)
182 //              return;

184         mask = get_mask(offset);
185         if (mask <= array_size)
186                 return;

188         conditions = get_conditions(offset);

190         name = expr_to_str(array_expr);
191         sm_warning("potential spectre issue '%s' [%s]%s",
192                 name,
```



```
193                 is_read(expr) ? "r" : "w",
194                 conditions ? " (local cap)" : "");
195         if (suppress_multiple)
196                 add_ignore_expr(my_id, array_expr);
197         free_string(name);
198 }

200 void check_spectre(int id)
201 {
202         my_id = id;

204         suppress_multiple = getenv("FULL_SPECTRE") == NULL;

206         if (option_project != PROJ_KERNEL)
207                 return;

209         add_hook(&array_check, OP_HOOK);
210 }
```

```
**********************************************************
    1718 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_sprintf_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static void match_sprintf(const char *fn, struct expression *expr, void *unused)
{
        struct expression *dest;
        struct expression *format_string;
        struct expression *data;
        char *data_name = NULL;
        int dest_size;
        char *format;
        int data_size;

        dest = get_argument_from_call_expr(expr->args, 0);
        format_string = get_argument_from_call_expr(expr->args, 1);
        data = get_argument_from_call_expr(expr->args, 2);

        dest_size = get_array_size_bytes(dest);
        if (!dest_size)
                return;
        format = expr_to_var(format_string);
        if (!format)
                return;
        if (strcmp(format, "\"%s\""))
                goto free;
        data_name = expr_to_str(data);
        data_size = get_size_from_strlen(data);
        if (!data_size)
                data_size = get_array_size_bytes(data);
        if (dest_size < data_size)
                sm_error("sprintf() copies too much data from '%s': %d vs %d",
                        data_name, data_size, dest_size);
free:
        free_string(data_name);
        free_string(format);
}

void check_sprintf_overflow(int id)
{
        add_function_hook("sprintf", &match_sprintf, NULL);
}
```

```
**********************************************************
    2093 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_stack.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The kernel has a small stack so putting huge structs and arrays on the
 * stack is a bug.
 *
 */

#include "smatch.h"

static int my_id;

static int total_size;
static int max_size;
static int max_lineno;
static int complained;

#define MAX_ALLOWED 1000

static void scope_end(void *_size)
{
        int size = PTR_INT(_size);
        total_size -= size;
}

static void match_declarations(struct symbol *sym)
{
        struct symbol *base;
        const char *name;

        base = get_base_type(sym);
        if (sym->ctype.modifiers & MOD_STATIC)
                return;
        name = sym->ident->name;
        total_size += type_bytes(base);
        if (total_size > max_size) {
                max_size = total_size;
                max_lineno = get_lineno();
        }
        if (type_bytes(base) >= MAX_ALLOWED) {
                complained = 1;
                sm_warning("'%s' puts %d bytes on stack", name, type_bytes(base)
        }
        add_scope_hook(&scope_end, INT_PTR(type_bytes(base)));
}
```



```
static void match_end_func(struct symbol *sym)
{
        if (__inline_fn)
                return;

        if ((max_size >= MAX_ALLOWED) && !complained) {
                sm_printf("%s:%d %s() ", get_filename(), max_lineno, get_functio
                sm_printf("warn: function puts %d bytes on stack\n", max_size);
        }
        total_size = 0;
        complained = 0;
        max_size = 0;
        max_lineno = 0;
}

void check_stack(int id)
{
        if (option_project != PROJ_KERNEL || !option_spammy)
                return;

        my_id = id;
        add_hook(&match_declarations, DECLARATION_HOOK);
        add_hook(&match_end_func, END_FUNC_HOOK);
}
```

```
**********************************************************
    1840 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_strcpy_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static void match_strcpy(const char *fn, struct expression *expr, void *unused)
{
        struct expression *dest;
        struct expression *data;
        char *dest_name = NULL;
        char *data_name = NULL;
        int dest_size;
        int data_size;

        dest = get_argument_from_call_expr(expr->args, 0);
        data = get_argument_from_call_expr(expr->args, 1);
        dest_size = get_array_size_bytes(dest);
        if (!dest_size)
                return;

        data_size = get_size_from_strlen(data);
        if (!data_size)
                data_size = get_array_size_bytes(data);

        /* If the size of both arrays is known and the destination
         * buffer is larger than the source buffer, we're okay.
         */
        if (data_size && dest_size >= data_size)
                return;

        dest_name = expr_to_str(dest);
        data_name = expr_to_str(data);

        if (data_size)
                sm_error("%s() '%s' too large for '%s' (%d vs %d)",
                         fn, data_name, dest_name, data_size, dest_size);
        else if (option_spammy)
                sm_warning("%s() '%s' of unknown size might be too large for '%s
                         fn, data_name, dest_name);

        free_string(dest_name);
        free_string(data_name);
}

void check_strcpy_overflow(int id)
{
```



```
        add_function_hook("strcpy", &match_strcpy, NULL);
}
```

```
**********************************************************
    4870 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_string_len.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This tries to find buffer overflows in sprintf().
 * I'll freely admit that the code is sort of crap.
 * Also if it sees "sprintf("%2d\n", x)" then it assumes x is less than 99.
 * That might not be true so there maybe buffer overflows which are missed.
 *
 */

#include <ctype.h>
#include "smatch.h"

static int my_id;

struct param_info {
        int buf_or_limit;
        int string;
};

struct param_info zero_one = {0, 1};

static int handle_format(struct expression *call, char **pp, int *arg_nr)
{
        struct expression *arg;
        char *p = *pp;
        int ret = 1;
        char buf[256];
        sval_t max;

        p++;  /* we passed it with *p == '%' */

        if (*p == '%') {
                p++;
                ret = 1;
                goto out_no_arg;
        }
        if (*p == 'c') {
                p++;
                ret = 1;
                goto out;
        }


        if (isdigit(*p) || *p == '.') {
```



```
                unsigned long num;

                if (*p == '.')
                        p++;

                num = strtoul(p, &p, 10);
                ret = num;

                while (*p == 'l')
                        p++;
                p++; /* eat the 'd' char */
                goto out;
        }

        if (*p == 'l') {
                p++;
                if (*p == 'l')
                        p++;
        }

        if (option_project == PROJ_KERNEL && *p == 'z')
                p++;

        if (option_project == PROJ_KERNEL && *p == 'p') {
                if (*(p + 1) == 'I' || *(p + 1) == 'i') {
                        char *eye;

                        eye = p + 1;
                        p += 2;
                        if (*p == 'h' || *p == 'n' || *p == 'b' || *p == 'l')
                                p++;
                        if (*p == '4') {
                                p++;
                                ret = 15;
                                goto out;
                        }
                        if (*p == '6') {
                                p++;
                                if (*p == 'c')
                                        p++;
                                if (*eye == 'I')
                                        ret = 39;
                                if (*eye == 'i')
                                        ret = 32;
                                goto out;
                        }
                }
                if (*(p + 1) == 'M') {
                        p += 2;
                        if (*p == 'R' || *p == 'F')
                                p++;
                        ret = 17;
                        goto out;
                }
                if (*(p + 1) == 'm') {
                        p += 2;
                        if (*p == 'R')
                                p++;
                        ret = 12;
                        goto out;
                }
        }

        arg = get_argument_from_call_expr(call->args, *arg_nr);
        if (!arg)
                goto out;
```

```
128         if (*p == 's') {
129                 ret = get_array_size_bytes(arg);
130                 if (ret < 0)
131                         ret = 1;
132                 /* we don't print the NUL here */
133                 ret--;
134                 p++;
135                 goto out;
136         }
137
138         if (*p != 'd' && *p != 'i' && *p != 'x' && *p != 'X' && *p != 'u' && *p
139                 ret = 1;
140                 p++;
141                 goto out;
142         }
143
144         get_absolute_max(arg, &max);
145
146         if (*p == 'x' || *p == 'X' || *p == 'p') {
147                 ret = snprintf(buf, sizeof(buf), "%llx", max.uvalue);
148         } else if (*p == 'u') {
149                 ret = snprintf(buf, sizeof(buf), "%llu", max.uvalue);
150         } else if (!expr_unsigned(arg)) {
151                 sval_t min;
152                 int tmp;
153
154                 ret = snprintf(buf, sizeof(buf), "%lld", max.value);
155                 get_absolute_min(arg, &min);
156                 tmp = snprintf(buf, sizeof(buf), "%lld", min.value);
157                 if (tmp > ret)
158                         ret = tmp;
159         } else {
160                 ret = snprintf(buf, sizeof(buf), "%lld", max.value);
161         }
162         p++;
163
164 out:
165         (*arg_nr)++;
166 out_no_arg:
167         *pp = p;
168         return ret;
169 }
170
171 int get_formatted_string_size(struct expression *call, int arg)
172 {
173         struct expression *expr;
174         char *p;
175         int count;
176
177         expr = get_argument_from_call_expr(call->args, arg);
178         if (!expr || expr->type != EXPR_STRING)
179                 return -1;
180
181         arg++;
182         count = 0;
183         p = expr->string->data;
184         while (*p) {
185
186                 if (*p == '%') {
187                         count += handle_format(call, &p, &arg);
188                 } else if (*p == '\\') {
189                         p++;
190                 }else {
191                         p++;
192                         count++;
```



```
193                 }
194         }
195
196         count++; /* count the NUL terminator */
197         return count;
198 }
199
200 static void match_not_limited(const char *fn, struct expression *call, void *inf
201 {
202         struct param_info *params = info;
203         struct range_list *rl;
204         struct expression *dest;
205         struct expression *arg;
206         int buf_size, size;
207         int user = 0;
208         int i;
209         int offset = 0;
210
211         dest = get_argument_from_call_expr(call->args, params->buf_or_limit);
212         dest = strip_expr(dest);
213         if (dest->type == EXPR_BINOP && dest->op == '+') {
214                 sval_t max;
215
216                 if (get_hard_max(dest->right, &max))
217                         offset = max.value;
218                 dest = dest->left;
219         }
220
221
222         buf_size = get_array_size_bytes(dest);
223         if (buf_size <= 0)
224                 return;
225
226         size = get_formatted_string_size(call, params->string);
227         if (size <= 0)
228                 return;
229         if (size < offset)
230                 size -= offset;
231         if (size <= buf_size)
232                 return;
233
234         i = 0;
235         FOR_EACH_PTR(call->args, arg) {
236                 if (i++ <= params->string)
237                         continue;
238                 if (get_user_rl(arg, &rl))
239                         user = 1;
240         } END_FOR_EACH_PTR(arg);
241
242         sm_error("format string overflow. buf_size: %d length: %d%s",
243                 buf_size, size, user ? " [user data]": "");
244 }
245
246 void check_string_len(int id)
247 {
248         my_id = id;
249         add_function_hook("sprintf", &match_not_limited, &zero_one);
250 }
```

```
**********************************************************
    1934 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_struct_type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_assign(const char *fn, struct expression *expr, void *_size_ar
{
        int size_arg = PTR_INT(_size_arg);
        struct expression *left;
        struct expression *call;
        struct expression *arg;
        struct symbol *left_type;
        struct symbol *size_type;

        left = strip_expr(expr->left);
        left_type = get_type(left);
        if (!left_type || left_type->type != SYM_PTR)
                return;
        left_type = get_real_base_type(left_type);
        if (!left_type || left_type->type != SYM_STRUCT)
                return;

        call = strip_expr(expr->right);
        arg = get_argument_from_call_expr(call->args, size_arg);
        if (!arg || arg->type != EXPR_SIZEOF || !arg->cast_type)
                return;
        size_type = arg->cast_type;
        if (size_type->type != SYM_NODE)
                return;
        size_type = get_real_base_type(size_type);
        if (size_type->type != SYM_STRUCT)
                return;
        if (strcmp(left_type->ident->name, size_type->ident->name) == 0)
                return;
        sm_warning("struct type mismatch '%s vs %s'", left_type->ident->name,
               size_type->ident->name);

}

void check_struct_type(int id)
{
        my_id = id;

        if (option_project == PROJ_KERNEL) {
```



```
                add_function_assign_hook("kmalloc", &match_assign, INT_PTR(0));
                add_function_assign_hook("kzalloc", &match_assign, INT_PTR(0));
        }
}
```

```
**********************************************************
    3558 Fri Dec 21 15:00:08 2018
new/usr/src/tools/smatch/src/check_syscall_arg_type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is to help create Trinity fuzzer templates.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(ARG_FD);
#if 0
STATE(arg_range);
STATE(arg_op);
STATE(arg_list);
STATE(arg_cpu);
STATE(arg_pathname);
#endif
// nr_segs * sizeof(struct iovec)
// if (nr_segs > UIO_MAXIOV)
#if 0
STATE(arg_ioveclen);
STATE(arg_sockaddrlen);
STATE(arg_socketinfo);
#endif

struct smatch_state *merge_states(struct smatch_state *s1, struct smatch_state *
{
        if (s1 == &undefined)
                return s2;
        return s1;
}

struct typedef_lookup {
        const char *name;
        struct symbol *sym;
        int failed;
};

static struct symbol *_typedef_lookup(const char *name)
{
        struct ident *id;
        struct symbol *node;
```



```
        id = built_in_ident(name);
        if (!id)
                return NULL;
        node = lookup_symbol(id, NS_TYPEDEF);
        if (!node || node->type != SYM_NODE)
                return NULL;
        return get_real_base_type(node);
}

static void typedef_lookup(struct typedef_lookup *tl)
{
        if (tl->sym || tl->failed)
                return;
        tl->sym = _typedef_lookup(tl->name);
        if (!tl->sym)
                tl->failed = 1;
}

static int is_mode_t(struct symbol *sym)
{
        static struct typedef_lookup umode_t = { .name = "umode_t" };
        struct symbol *type;

        typedef_lookup(&umode_t);
        if (!umode_t.sym)
                return 0;
        type = get_base_type(sym);
        if (type == umode_t.sym)
                return 1;
        return 0;
}

static int is_pid_t(struct symbol *sym)
{
        static struct typedef_lookup pid_t = { .name = "pid_t" };
        struct symbol *type;

        typedef_lookup(&pid_t);
        if (!pid_t.sym)
                return 0;
        type = get_base_type(sym);
        if (type == pid_t.sym)
                return 1;
        return 0;
}

static const char *get_arg_type_from_type(struct symbol *sym)
{
        struct symbol *type;

        if (is_mode_t(sym))
                return "ARG_MODE_T";
        if (is_pid_t(sym))
                return "ARG_PID";

        type = get_real_base_type(sym);
        if (!type || type->type != SYM_PTR)
                return NULL;
        type = get_real_base_type(type);
        if (!type)
                return NULL;
        if (type == &char_ctype)
                return "ARG_MMAP";
        if (!type->ident)
                return NULL;
```

```
127         if (strcmp(type->ident->name, "iovec") == 0)
128                 return "ARG_IOVEC";
129         if (strcmp(type->ident->name, "sockaddr") == 0)
130                 return "ARG_SOCKADDR";
131         return "ARG_ADDRESS";
132 }

134 static void match_fdget(const char *fn, struct expression *expr, void *unused)
135 {
136         struct expression *arg;

138         arg = get_argument_from_call_expr(expr->args, 0);
139         set_state_expr(my_id, arg, &ARG_FD);
140 }

142 const char *get_syscall_arg_type(struct symbol *sym)
143 {
144         struct smatch_state *state;
145         const char *type;

147         if (!sym || !sym->ident)
148                 return "ARG_UNDEFINED";
149         type = get_arg_type_from_type(sym);
150         if (type)
151                 return type;
152         state = get_state(my_id, sym->ident->name, sym);
153         if (!state)
154                 return "ARG_UNDEFINED";
155         return state->name;
156 }

158 void check_syscall_arg_type(int id)
159 {
160         my_id = id;
161         if (option_project != PROJ_KERNEL)
162                 return;

164         add_merge_hook(my_id, &merge_states);
165         add_function_hook("fdget", &match_fdget, NULL);
166 }
```

```
**********************************************************
    3080 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_template.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 20XX Your Name.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * First of all, it's best if you lower your expectations from finding
 * errors to just finding suspicious code.  There tends to be a lot
 * of false positives so having low expectations helps.
 *
 * For this test let's look for functions that return a negative value
 * with a semaphore held.
 *
 * This is just a template check.  It's designed for teaching
 * only and is deliberately less useful than it could be.  check_locking.c
 * is a better real world test.
 *
 * The biggest short coming is that it assumes a function isn't supposed
 * to return negative with a lock held.  Also it assumes the function was
 * called without the lock held. It would be better if it handled the stuff
 * like this:
 *     ret = -ENOMEM;
 *     return ret;
 * Another idea would be to test other kinds of locks besides just semaphores.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

STATE(lock);
STATE(unlock);

/*
 * unmatched_state() deals with the case where code is known to be
 * locked on one path but not known on the other side of a merge.  Here
 * we assume it's the opposite.
 */

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        if (sm->state == &lock)
                return &unlock;
        if (sm->state == &unlock)
                return &lock;
        return &undefined;
}
```



```
}

static void match_call(struct expression *expr)
{
        char *fn_name;
        struct expression *sem_expr;
        char *sem_name;

        fn_name = expr_to_var(expr->fn);
        if (!fn_name || (strcmp(fn_name, "down") && strcmp(fn_name, "up")))
                goto free_fn;

        sem_expr = get_argument_from_call_expr(expr->args, 0);
        sem_name = expr_to_var(sem_expr);
        if (!strcmp(fn_name, "down")) {
                set_state(my_id, sem_name, NULL, &lock);
        } else {
                set_state(my_id, sem_name, NULL, &unlock);
        }
        free_string(sem_name);
free_fn:
        free_string(fn_name);
}

static void match_return(struct expression *ret_value)
{
        sval_t ret_val;
        struct stree *stree;
        struct sm_state *tmp;

        if (!get_value(ret_value, &ret_val) || sval_cmp_val(ret_val, 0) >= 0)
                return;

        stree = __get_cur_stree();
        FOR_EACH_MY_SM(my_id, stree, tmp) {
                if (tmp->state != &unlock)
                        sm_warning("returned negative with %s semaphore held",
                                   tmp->name);
        } END_FOR_EACH_SM(tmp);
}

void check_template(int id)
{
        my_id = id;
        add_unmatched_state_hook(my_id, &unmatched_state);
        add_hook(&match_call, FUNCTION_CALL_HOOK);
        add_hook(&match_return, RETURN_HOOK);
}
```

```
**********************************************************
    1649 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_test_bit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static void match_test_bit(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg;
        char *macro;

        arg = get_argument_from_call_expr(expr->args, 0);
        arg = strip_expr(arg);

        if (!arg || arg->type != EXPR_BINOP)
                return;
        if (arg->op != '|' && arg->op != SPECIAL_LEFTSHIFT)
                return;
        macro = get_macro_name(arg->pos);
        if (macro && strstr(macro, "cpu_has"))
                return;
        sm_warning("test_bit() takes a bit number");
}

void check_test_bit(int id)
{
        my_id = id;

        if (option_project != PROJ_KERNEL)
                return;

        add_function_hook("test_bit", &match_test_bit, NULL);
        add_function_hook("variable_test_bit", &match_test_bit, NULL);
        add_function_hook("set_bit", &match_test_bit, NULL);
        add_function_hook("clear_bit", &match_test_bit, NULL);
        add_function_hook("test_and_clear_bit", &match_test_bit, NULL);
        add_function_hook("test_and_set_bit", &match_test_bit, NULL);
}
```

```
**********************************************************
    3192 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_testing_index_after_use.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

/*
 * This check has two smatch IDs.
 * my_used_id - keeps a record of array offsets that have been used.
 *              If the code checks that they are within bounds later on,
 *              we complain about using an array offset before checking
 *              that it is within bounds.
 */
static int my_used_id;

static void delete(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_used_id, sm->name, sm->sym, &undefined);
}

static int get_the_max(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;

        if (get_hard_max(expr, sval))
                return 1;
        if (!option_spammy)
                return 0;
        if (get_fuzzy_max(expr, sval))
                return 1;
        if (get_user_rl(expr, &rl)) {
                *sval = rl_max(rl);
                return 1;
        }
        return 0;
}

static void array_check(struct expression *expr)
{
        struct expression *array_expr;
        int array_size;
        struct expression *offset;
        sval_t max;
```



```
        expr = strip_expr(expr);
        if (!is_array(expr))
                return;

        array_expr = get_array_base(expr);
        array_size = get_array_size(array_expr);
        if (!array_size || array_size == 1)
                return;

        offset = get_array_offset(expr);
        if (!get_the_max(offset, &max)) {
                if (getting_address())
                        return;
                if (is_capped(offset))
                        return;
                set_state_expr(my_used_id, offset, alloc_state_num(array_size));
        }
}

static void match_condition(struct expression *expr)
{
        int left;
        sval_t sval;
        struct state_list *slist;
        struct sm_state *tmp;
        int boundary;

        if (!expr || expr->type != EXPR_COMPARE)
                return;
        if (get_macro_name(expr->pos))
                return;
        if (get_implied_value(expr->left, &sval))
                left = 1;
        else if (get_implied_value(expr->right, &sval))
                left = 0;
        else
                return;

        if (left)
                slist = get_possible_states_expr(my_used_id, expr->right);
        else
                slist = get_possible_states_expr(my_used_id, expr->left);
        if (!slist)
                return;
        FOR_EACH_PTR(slist, tmp) {
                if (tmp->state == &merged || tmp->state == &undefined)
                        continue;
                boundary = PTR_INT(tmp->state->data);
                boundary -= sval.value;
                if (boundary < 1 && boundary > -1) {
                        char *name;

                        name = expr_to_var(left ? expr->right : expr->left);
                        sm_error("testing array offset '%s' after use.", name);
                        return;
                }
        } END_FOR_EACH_PTR(tmp);
}

void check_testing_index_after_use(int id)
{
        my_used_id = id;
        add_hook(&array_check, OP_HOOK);
        add_hook(&match_condition, CONDITION_HOOK);
        add_modification_hook(my_used_id, &delete);
```

```
127 }
```

```
**********************************************************
    2491 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_trinity_generator.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The idea is to generate syscall templates for the Trinity fuzzer.  There
 * isn't currently quite enough information to do it right but I want to start
 * and see how far I can get.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

FILE *sysc_fd;

static int gen_custom_struct(int nr, struct symbol *arg)
{
        return 0;
}

static void print_arg(int nr, struct symbol *arg)
{
        fprintf(sysc_fd, "\t.arg%dname = \"%s\",\n", nr + 1, arg->ident->name);
        fprintf(sysc_fd, "\t.arg%dtype = %s,\n", nr + 1, get_syscall_arg_type(ar
}

static void match_return(struct expression *ret_value)
{
        struct symbol *arg;
        int num_args;
        char *name;
        int i;
        char buf[256];
        int has_custom_struct[6];

        if (!get_function() || !cur_func_sym)
                return;
        if (strncmp(get_function(), "SYSC_", 5) != 0)
                return;

        num_args = ptr_list_size((struct ptr_list *)cur_func_sym->ctype.base_typ
        name = get_function() + 5;

        snprintf(buf, sizeof(buf), "smatch_trinity_%s", name);
```



```
        sysc_fd = fopen(buf, "w");
        if (!sm_outfd) {
                printf("Error:  Cannot open %s\n", buf);
                return;
        }

        i = 0;
        FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
                if (gen_custom_struct(i, arg))
                        has_custom_struct[i] = true;
                else
                        has_custom_struct[i] = false;
                i++;
        } END_FOR_EACH_PTR(arg);

        fprintf(sysc_fd, "struct syscallentry sm_%s = {\n", name);
        fprintf(sysc_fd, "\t.name = \"%s\",\n", name);
        fprintf(sysc_fd, "\t.num_args = %d,\n", num_args);

        i = 0;
        FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
                if (has_custom_struct[i])
                        ;
                else
                        print_arg(i++, arg);
        } END_FOR_EACH_PTR(arg);

        fprintf(sysc_fd, "};\n");
}

void check_trinity_generator(int id)
{
        my_id = id;

        if (option_project != PROJ_KERNEL)
                return;
        add_hook(&match_return, RETURN_HOOK);
}
```

```
**********************************************************
    1681 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static int in_function(const char *fn)
{
        char *cur_func = get_function();

        if (!cur_func)
                return 0;
        if (!strcmp(cur_func, fn))
                return 1;
        return 0;
}

static void match_free(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg_expr;
        char *name;
        struct symbol *type;

        arg_expr = get_argument_from_call_expr(expr->args, 0);
        type = get_pointer_type(arg_expr);
        if (!type || !type->ident)
                return;

        name = expr_to_str(arg_expr);

        if (!strcmp("sk_buff", type->ident->name)) {
                sm_error("use kfree_skb() here instead of kfree(%s)", name);
        } else if (!strcmp("net_device", type->ident->name)) {
                if (in_function("alloc_netdev"))
                        return;
                if (in_function("alloc_netdev_mqs"))
                        return;
                sm_error("use free_netdev() here instead of kfree(%s)", name);
        }

        free_string(name);
}

void check_type(int id)
{
```



```
        my_id = id;
        if (option_project == PROJ_KERNEL)
                add_function_hook("kfree", &match_free, NULL);
}
```

```
**********************************************************
    8358 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_uninitialized.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(uninitialized);
STATE(initialized);

static void pre_merge_hook(struct sm_state *sm)
{
        if (is_impossible_path())
                set_state(my_id, sm->name, sm->sym, &initialized);
}

static void mark_members_uninitialized(struct symbol *sym)
{
        struct symbol *struct_type, *tmp, *base_type;
        char buf[256];

        struct_type = get_real_base_type(sym);
        FOR_EACH_PTR(struct_type->symbol_list, tmp) {
                if (!tmp->ident)
                        continue;
                base_type = get_real_base_type(tmp);
                if (!base_type ||
                    base_type->type == SYM_STRUCT ||
                    base_type->type == SYM_ARRAY ||
                    base_type->type == SYM_UNION)
                        continue;
                snprintf(buf, sizeof(buf), "%s.%s", sym->ident->name, tmp->ident
                set_state(my_id, buf, sym, &uninitialized);
        } END_FOR_EACH_PTR(tmp);
}

static void match_declarations(struct symbol *sym)
{
        struct symbol *type;

        if (sym->initializer)
                return;

        type = get_real_base_type(sym);
```



```
        /* Smatch is crap at tracking arrays */
        if (type->type == SYM_ARRAY)
                return;
        if (type->type == SYM_UNION)
                return;
        if (sym->ctype.modifiers & MOD_STATIC)
                return;

        if (!sym->ident)
                return;

        if (type->type == SYM_STRUCT) {
                mark_members_uninitialized(sym);
                return;
        }

        set_state(my_id, sym->ident->name, sym, &uninitialized);
}

static void extra_mod_hook(const char *name, struct symbol *sym, struct expressi
{
        if (!sym || !sym->ident)
                return;
        if (strcmp(name, sym->ident->name) != 0)
                return;
        set_state(my_id, name, sym, &initialized);
}

static void match_assign(struct expression *expr)
{
        struct expression *right;

        right = strip_expr(expr->right);
        if (right->type == EXPR_PREOP && right->op == '&')
                set_state_expr(my_id, right->unop, &initialized);
}

static int is_initialized(struct expression *expr)
{
        struct sm_state *sm;

        expr = strip_expr(expr);
        if (expr->type != EXPR_SYMBOL)
                return 1;
        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return 1;
        if (!slist_has_state(sm->possible, &uninitialized))
                return 1;
        return 0;
}

static void match_dereferences(struct expression *expr)
{
        char *name;

        if (parse_error)
                return;

        if (expr->type != EXPR_PREOP)
                return;
        if (is_impossible_path())
                return;
        if (is_initialized(expr->unop))
                return;
```

```
127         name = expr_to_str(expr->unop);
128         sm_error("potentially dereferencing uninitialized '%s'.", name);
129         free_string(name);
130
131         set_state_expr(my_id, expr->unop, &initialized);
132 }
133
134 static void match_condition(struct expression *expr)
135 {
136         char *name;
137
138         if (parse_error)
139                 return;
140
141         if (is_impossible_path())
142                 return;
143
144         if (is_initialized(expr))
145                 return;
146
147         name = expr_to_str(expr);
148         sm_error("potentially using uninitialized '%s'.", name);
149         free_string(name);
150
151         set_state_expr(my_id, expr, &initialized);
152 }
153
154 static void match_call(struct expression *expr)
155 {
156         struct expression *arg;
157         char *name;
158
159         if (parse_error)
160                 return;
161
162         if (is_impossible_path())
163                 return;
164
165         FOR_EACH_PTR(expr->args, arg) {
166                 if (is_initialized(arg))
167                         continue;
168
169                 name = expr_to_str(arg);
170                 sm_warning("passing uninitialized '%s'", name);
171                 free_string(name);
172
173                 set_state_expr(my_id, arg, &initialized);
174         } END_FOR_EACH_PTR(arg);
175 }
176
177 static int param_used_callback(void *found, int argc, char **argv, char **azColN
178 {
179         *(int *)found = 1;
180         return 0;
181 }
182
183 static int member_is_used(struct expression *call, int param, char *printed_name
184 {
185         int found;
186
187         /* for function pointers assume everything is used */
188         if (call->fn->type != EXPR_SYMBOL)
189                 return 0;
190
191         found = 0;
192         run_sql(&param_used_callback, &found,
```



```
193                 "select * from return_implies where %s and type = %d and paramet
194                 get_static_filter(call->fn->symbol), PARAM_USED, param, printed_
195         return found;
196 }
197
198 static void match_call_struct_members(struct expression *expr)
199 {
200         struct symbol *type, *sym;
201         struct expression *arg;
202         struct sm_state *sm;
203         char *arg_name;
204         char buf[256];
205         int param;
206
207         return;
208
209         if (parse_error)
210                 return;
211
212         param = -1;
213         FOR_EACH_PTR(expr->args, arg) {
214                 param++;
215                 if (arg->type != EXPR_PREOP || arg->op != '&')
216                         continue;
217                 type = get_type(arg->unop);
218                 if (!type || type->type != SYM_STRUCT)
219                         continue;
220                 arg_name = expr_to_var_sym(arg->unop, &sym);
221                 if (!arg_name || !sym)
222                         goto free;
223                 FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
224                         if (sm->sym != sym)
225                                 continue;
226                         if (!slist_has_state(sm->possible, &uninitialized))
227                                 continue;
228                         snprintf(buf, sizeof(buf), "$->%s", sm->name + strlen(ar
229                         if (!member_is_used(expr, param, buf))
230                                 goto free;
231                         sm_warning("struct member %s is uninitialized", sm->name
232                 } END_FOR_EACH_SM(sm);
233
234 free:
235                 free_string(arg_name);
236         } END_FOR_EACH_PTR(arg);
237 }
238
239 static int is_being_modified(struct expression *expr)
240 {
241         struct expression *parent;
242         struct statement *stmt;
243
244         parent = expr_get_parent_expr(expr);
245         if (!parent)
246                 return 0;
247         while (parent->type == EXPR_PREOP && parent->op == '(') {
248                 parent = expr_get_parent_expr(parent);
249                 if (!parent)
250                         return 0;
251         }
252         if (parent->type == EXPR_PREOP && parent->op == '&')
253                 return 1;
254         if (parent->type == EXPR_ASSIGNMENT && expr_equiv(parent->left, expr))
255                 return 1;
256
257         stmt = last_ptr_list((struct ptr_list *)big_statement_stack);
258         if (stmt && stmt->type == STMT_ASM)
```

```
259                 return 1;

261         return 0;
262 }

264 static void match_symbol(struct expression *expr)
265 {
266         char *name;

268         if (parse_error)
269                 return;

271         if (is_impossible_path())
272                 return;

274         if (is_initialized(expr))
275                 return;

277         if (is_being_modified(expr))
278                 return;

280         name = expr_to_str(expr);
281         sm_error("uninitialized symbol '%s'.", name);
282         free_string(name);

284         set_state_expr(my_id, expr, &initialized);
285 }

287 static void match_untracked(struct expression *call, int param)
288 {
289         struct expression *arg;

291         arg = get_argument_from_call_expr(call->args, param);
292         arg = strip_expr(arg);
293         if (!arg || arg->type != EXPR_PREOP || arg->op != '&')
294                 return;
295         arg = strip_expr(arg->unop);
296         set_state_expr(my_id, arg, &initialized);
297 }

299 static void match_ignore_param(const char *fn, struct expression *expr, void *_a
300 {
301         int arg_nr = PTR_INT(_arg_nr);
302         struct expression *arg;

304         arg = get_argument_from_call_expr(expr->args, arg_nr);
305         arg = strip_expr(arg);
306         if (!arg)
307                 return;
308         if (arg->type != EXPR_PREOP || arg->op != '&')
309                 return;
310         arg = strip_expr(arg->unop);
311         set_state_expr(my_id, arg, &initialized);
312 }

314 static void register_ignored_params_from_file(void)
315 {
316         char name[256];
317         struct token *token;
318         const char *func;
319         char prev_func[256];
320         int param;

322         memset(prev_func, 0, sizeof(prev_func));
323         snprintf(name, 256, "%s.ignore_uninitialized_param", option_project_str)
324         name[255] = '\0';
```



```
325         token = get_tokens_file(name);
326         if (!token)
327                 return;
328         if (token_type(token) != TOKEN_STREAMBEGIN)
329                 return;
330         token = token->next;
331         while (token_type(token) != TOKEN_STREAMEND) {
332                 if (token_type(token) != TOKEN_IDENT)
333                         return;
334                 func = show_ident(token->ident);

336                 token = token->next;
337                 if (token_type(token) != TOKEN_NUMBER)
338                         return;
339                 param = atoi(token->number);

341                 add_function_hook(func, &match_ignore_param, INT_PTR(param));

343                 token = token->next;
344         }
345         clear_token_alloc();
346 }

348 void check_uninitialized(int id)
349 {
350         my_id = id;

352         add_hook(&match_declarations, DECLARATION_HOOK);
353         add_extra_mod_hook(&extra_mod_hook);
354         add_hook(&match_assign, ASSIGNMENT_HOOK);
355         add_untracked_param_hook(&match_untracked);
356         add_pre_merge_hook(my_id, &pre_merge_hook);

358         add_hook(&match_dereferences, DEREF_HOOK);
359         add_hook(&match_condition, CONDITION_HOOK);
360         add_hook(&match_call, FUNCTION_CALL_HOOK);
361         add_hook(&match_call_struct_members, FUNCTION_CALL_HOOK);
362         add_hook(&match_symbol, SYM_HOOK);

364         register_ignored_params_from_file();
365 }
```

```
**********************************************************
    6586 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_unreachable.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static int print_unreached = 1;
static struct string_list *turn_off_names;
static struct string_list *ignore_names;

static int empty_statement(struct statement *stmt)
{
        if (!stmt)
                return 0;
        if (stmt->type == STMT_EXPRESSION && !stmt->expression)
                return 1;
        return 0;
}

static int is_last_stmt(struct statement *cur_stmt)
{
        struct symbol *fn = get_base_type(cur_func_sym);
        struct statement *stmt;

        if (!fn)
                return 0;
        stmt = fn->stmt;
        if (!stmt)
                stmt = fn->inline_stmt;
        if (!stmt || stmt->type != STMT_COMPOUND)
                return 0;
        stmt = last_ptr_list((struct ptr_list *)stmt->stmts);
        if (stmt == cur_stmt)
                return 1;
        return 0;
}

static void print_unreached_initializers(struct symbol_list *sym_list)
{
        struct symbol *sym;

        FOR_EACH_PTR(sym_list, sym) {
                if (sym->initializer && !(sym->ctype.modifiers & MOD_STATIC))
                        sm_msg("info: '%s' is not actually initialized (unreache
                                (sym->ident ? sym->ident->name : "this variable"
        } END_FOR_EACH_PTR(sym);
}

static int is_ignored_macro(struct statement *stmt)
{
        char *name;
        char *tmp;

        name = get_macro_name(stmt->pos);
        if (!name)
                return 0;

        FOR_EACH_PTR(ignore_names, tmp) {
                if (strcmp(tmp, name) == 0)
                        return 1;
        } END_FOR_EACH_PTR(tmp);

        return 0;
}

static int prev_line_was_endif(struct statement *stmt)
{
        struct token *token;
        struct position pos = stmt->pos;

        pos.line--;
        pos.pos = 2;

        token = pos_get_token(pos);
        if (token && token_type(token) == TOKEN_IDENT &&
            strcmp(show_ident(token->ident), "endif") == 0)
                return 1;

        pos.line--;
        token = pos_get_token(pos);
        if (token && token_type(token) == TOKEN_IDENT &&
            strcmp(show_ident(token->ident), "endif") == 0)
                return 1;

        return 0;
}

static int we_jumped_into_the_middle_of_a_loop(struct statement *stmt)
{
        struct statement *prev;

        /*
         * Smatch doesn't handle loops correctly and this is a hack.  What we
         * do is that if the first unreachable statement is a loop and the
         * previous statement was a goto then it's probably code like this:
         *      goto first;
         *      for (;;) {
         *              frob();
         * first:
         *              more_frob();
         *      }
         * Every statement is reachable but only on the second iteration.
         */

        if (stmt->type != STMT_ITERATOR)
                return 0;
        prev = get_prev_statement();
        if (prev && prev->type == STMT_GOTO)
                return 1;
        return 0;
}
```

```
128 static void unreachable_stmt(struct statement *stmt)
129 {
131         if (__inline_fn)
132                 return;
134         if (!__path_is_null()) {
135                 print_unreached = 1;
136                 return;
137         }
139         /* if we hit a label then assume there is a matching goto */
140         if (stmt->type == STMT_LABEL)
141                 print_unreached = 0;
142         if (prev_line_was_endif(stmt))
143                 print_unreached = 0;
144         if (we_jumped_into_the_middle_of_a_loop(stmt))
145                 print_unreached = 0;
147         if (!print_unreached)
148                 return;
149         if (empty_statement(stmt))
150                 return;
151         if (is_ignored_macro(stmt))
152                 return;
154         switch (stmt->type) {
155         case STMT_COMPOUND: /* after a switch before a case stmt */
156         case STMT_RANGE:
157         case STMT_CASE:
158                 return;
159         case STMT_DECLARATION: /* switch (x) { int a; case foo: ... */
160                 print_unreached_initializers(stmt->declaration);
161                 return;
162         case STMT_RETURN: /* gcc complains if you don't have a return statement
163                 if (is_last_stmt(stmt))
164                         return;
165                 break;
166         case STMT_GOTO:
167                 /* people put extra breaks inside switch statements */
168                 if (stmt->goto_label && stmt->goto_label->type == SYM_NODE &&
169                     strcmp(stmt->goto_label->ident->name, "break") == 0)
170                         return;
171                 break;
172         default:
173                 break;
174         }
175         sm_msg("info: ignoring unreachable code.");
176         print_unreached = 0;
177 }
179 static int is_turn_off(char *name)
180 {
181         char *tmp;
183         if (!name)
184                 return 0;
186         FOR_EACH_PTR(turn_off_names, tmp) {
187                 if (strcmp(tmp, name) == 0)
188                         return 1;
189         } END_FOR_EACH_PTR(tmp);
191         return 0;
192 }
```



```
194 static char *get_function_name(struct statement *stmt)
195 {
196         struct expression *expr;
198         if (stmt->type != STMT_EXPRESSION)
199                 return NULL;
200         expr = stmt->expression;
201         if (!expr || expr->type != EXPR_CALL)
202                 return NULL;
203         if (expr->fn->type != EXPR_SYMBOL || !expr->fn->symbol_name)
204                 return NULL;
205         return expr->fn->symbol_name->name;
206 }
208 static void turn_off_unreachable(struct statement *stmt)
209 {
210         char *name;
212         name = get_macro_name(stmt->pos);
213         if (is_turn_off(name)) {
214                 print_unreached = 0;
215                 return;
216         }
218         if (stmt->type == STMT_IF &&
219             known_condition_true(stmt->if_conditional) &&  __path_is_null()) {
220                 print_unreached = 0;
221                 return;
222         }
224         name = get_function_name(stmt);
225         if (is_turn_off(name))
226                 print_unreached = 0;
227 }
229 static void register_turn_off_macros(void)
230 {
231         struct token *token;
232         char *macro;
233         char name[256];
235         if (option_project == PROJ_NONE)
236                 strcpy(name, "unreachable.turn_off");
237         else
238                 snprintf(name, 256, "%s.unreachable.turn_off", option_project_st
240         token = get_tokens_file(name);
241         if (!token)
242                 return;
243         if (token_type(token) != TOKEN_STREAMBEGIN)
244                 return;
245         token = token->next;
246         while (token_type(token) != TOKEN_STREAMEND) {
247                 if (token_type(token) != TOKEN_IDENT)
248                         return;
249                 macro = alloc_string(show_ident(token->ident));
250                 add_ptr_list(&turn_off_names, macro);
251                 token = token->next;
252         }
253         clear_token_alloc();
254 }
256 static void register_ignored_macros(void)
257 {
258         struct token *token;
```

```
259         char *macro;
260         char name[256];

262         if (option_project == PROJ_NONE)
263                 strcpy(name, "unreachable.ignore");
264         else
265                 snprintf(name, 256, "%s.unreachable.ignore", option_project_str)

267         token = get_tokens_file(name);
268         if (!token)
269                 return;
270         if (token_type(token) != TOKEN_STREAMBEGIN)
271                 return;
272         token = token->next;
273         while (token_type(token) != TOKEN_STREAMEND) {
274                 if (token_type(token) != TOKEN_IDENT)
275                         return;
276                 macro = alloc_string(show_ident(token->ident));
277                 add_ptr_list(&ignore_names, macro);
278                 token = token->next;
279         }
280         clear_token_alloc();
281 }

283 void check_unreachable(int id)
284 {
285         my_id = id;

287         register_turn_off_macros();
288         register_ignored_macros();
289         add_hook(&unreachable_stmt, STMT_HOOK);
290         add_hook(&turn_off_unreachable, STMT_HOOK_AFTER);
291 }
```

```
**********************************************************
    5349 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_unused_ret.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This check is supposed to find places like this:
 * err = foo();
 * err = bar();
 * if (err)
 *         return err;
 * (the first assignment isn't used)
 *
 * How the check works is that every assignment gets an ID.
 * We store that assignment ID in a list of assignments that
 * haven't been used.  We also set the state of 'err' from
 * the example above to be.  Then when we use 'err' we remove
 * it from the list.  At the end of the function we print
 * a list of assignments that still haven't been used.
 *
 * Note that this check only works for assignments to
 * EXPR_SYMBOL.  Maybe it could be modified to cover other
 * assignments later but then you would have to deal with
 * scope issues.
 *
 * Also this state is quite tied to the order the callbacks
 * are called in smatch_flow.c.  (If the order changed it
 * would break).
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_function_hashtable.h"

static int my_id;

struct assignment {
        int assign_id;
        char *name;
        char *function;
        int line;
};
ALLOCATOR(assignment, "assignment id");
DECLARE_PTR_LIST(assignment_list, struct assignment);
static struct assignment_list *assignment_list;

static struct expression *skip_this;
```



```
static int assign_id;

static DEFINE_HASHTABLE_INSERT(insert_func, char, int);
static DEFINE_HASHTABLE_SEARCH(search_func, char, int);
static struct hashtable *ignored_funcs;

static const char *kernel_ignored[] = {
        "inb",
        "inl",
        "inw",
        "readb",
        "readl",
        "readw",
};

static char *get_fn_name(struct expression *expr)
{
        if (expr->type != EXPR_CALL)
                return NULL;
        if (expr->fn->type != EXPR_SYMBOL)
                return NULL;
        return expr_to_var(expr->fn);
}

static int ignored_function(struct expression *expr)
{
        char *func;
        int ret = 0;

        func = get_fn_name(expr);
        if (!func)
                return 0;
        if (search_func(ignored_funcs, func))
                ret = 1;
        free_string(func);
        return ret;
}

static void match_assign_call(struct expression *expr)
{
        struct expression *left;
        struct assignment *assign;

        if (final_pass)
                return;
        if (in_condition())
                return;
        if (expr->op != '=')
                return;
        if (unreachable())
                return;
        if (ignored_function(expr->right))
                return;
        left = strip_expr(expr->left);
        if (!left || left->type != EXPR_SYMBOL)
                return;
        if (left->symbol->ctype.modifiers & (MOD_TOPLEVEL | MOD_EXTERN | MOD_STA
                return;

        skip_this = left;

        set_state_expr(my_id, left, alloc_state_num(assign_id));

        assign = __alloc_assignment(0);
        assign->assign_id = assign_id++;
        assign->name = expr_to_var(left);
```

```
127         assign->function = get_fn_name(expr->right);
128         assign->line = get_lineno();
129         add_ptr_list(&assignment_list, assign);
130 }

132 static void match_assign(struct expression *expr)
133 {
134         struct expression *left;

136         if (expr->op != '=')
137                 return;
138         left = strip_expr(expr->left);
139         if (!left || left->type != EXPR_SYMBOL)
140                 return;
141         set_state_expr(my_id, left, &undefined);
142 }

144 static void delete_used(int assign_id)
145 {
146         struct assignment *tmp;

148         FOR_EACH_PTR(assignment_list, tmp) {
149                 if (tmp->assign_id == assign_id) {
150                         DELETE_CURRENT_PTR(tmp);
151                         return;
152                 }
153         } END_FOR_EACH_PTR(tmp);
154 }

156 static void delete_used_symbols(struct state_list *possible)
157 {
158         struct sm_state *tmp;

160         FOR_EACH_PTR(possible, tmp) {
161                 delete_used(PTR_INT(tmp->state->data));
162         } END_FOR_EACH_PTR(tmp);
163 }

165 static void match_symbol(struct expression *expr)
166 {
167         struct sm_state *sm;

169         expr = strip_expr(expr);
170         if (expr == skip_this)
171                 return;
172         sm = get_sm_state_expr(my_id, expr);
173         if (!sm)
174                 return;
175         delete_used_symbols(sm->possible);
176         set_state_expr(my_id, expr, &undefined);
177 }

179 static void match_end_func(struct symbol *sym)
180 {
181         struct assignment *tmp;

183         if (__inline_fn)
184                 return;
185         FOR_EACH_PTR(assignment_list, tmp) {
186                 sm_printf("%s:%d %s() ", get_filename(), tmp->line, get_function
187                 sm_printf("warn: unused return: %s = %s()\n",
188                         tmp->name, tmp->function);
189         } END_FOR_EACH_PTR(tmp);
190 }

192 static void match_after_func(struct symbol *sym)
```



```
193 {
194         if (__inline_fn)
195                 return;
196         clear_assignment_alloc();
197         __free_ptr_list((struct ptr_list **)&assignment_list);
198 }

200 void check_unused_ret(int id)
201 {
202         my_id = id;

204         /* It turns out that this test is worthless unless you use --two-passes.
205         if (!option_two_passes)
206                 return;
207         add_hook(&match_assign_call, CALL_ASSIGNMENT_HOOK);
208         add_hook(&match_assign, ASSIGNMENT_HOOK);
209         add_hook(&match_symbol, SYM_HOOK);
210         add_hook(&match_end_func, END_FUNC_HOOK);
211         add_hook(&match_after_func, AFTER_FUNC_HOOK);
212         ignored_funcs = create_function_hashtable(100);
213         if (option_project == PROJ_KERNEL) {
214                 int i;

216                 for (i = 0; i < ARRAY_SIZE(kernel_ignored); i++)
217                         insert_func(ignored_funcs, (char *)kernel_ignored[i], (i
218         }
219 }
```

```
**********************************************************
    6846 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_unwind.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is a kernel check to make sure we unwind everything on
 * on errors.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

#define EBUSY 16
#define MAX_ERRNO 4095

static int my_id;

STATE(allocated);
STATE(unallocated);

/* state of unwind function */
STATE(called);

static int was_passed_as_param(struct expression *expr)
{
        char *name;
        struct symbol *sym;
        struct symbol *arg;

        name = expr_to_var_sym(expr, &sym);
        if (!name)
                return 0;
        free_string(name);

        FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
                if (arg == sym)
                        return 1;
        } END_FOR_EACH_PTR(arg);
        return 0;
}

static void print_unwind_functions(const char *fn, struct expression *expr, void
{
        struct expression *arg_expr;
        int arg_no = PTR_INT(_arg_no);
```



```
        static struct symbol *last_printed = NULL;

        arg_expr = get_argument_from_call_expr(expr->args, arg_no);
        if (!was_passed_as_param(arg_expr))
                return;
        if (last_printed == cur_func_sym)
                return;
        last_printed = cur_func_sym;
        sm_msg("info: is unwind function");
}

static void request_granted(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *_arg_no)
{
        struct expression *arg_expr;
        int arg_no = PTR_INT(_arg_no);

        if (arg_no == -1) {
                if (!assign_expr)
                        return;
                arg_expr = assign_expr->left;
        } else {
                arg_expr = get_argument_from_call_expr(call_expr->args, arg_no);
        }
        set_state_expr(my_id, arg_expr, &allocated);
}

static void request_denied(const char *fn, struct expression *call_expr,
                        struct expression *assign_expr, void *_arg_no)
{
        struct expression *arg_expr;
        int arg_no = PTR_INT(_arg_no);

        if (arg_no == -1) {
                if (!assign_expr)
                        return;
                arg_expr = assign_expr->left;
        } else {
                arg_expr = get_argument_from_call_expr(call_expr->args, arg_no);
        }
        set_state_expr(my_id, arg_expr, &unallocated);
}

static void match_release(const char *fn, struct expression *expr, void *_arg_no
{
        struct expression *arg_expr;
        int arg_no = PTR_INT(_arg_no);

        arg_expr = get_argument_from_call_expr(expr->args, arg_no);
        if (get_state_expr(my_id, arg_expr))
                set_state_expr(my_id, arg_expr, &unallocated);
        set_equiv_state_expr(my_id, arg_expr, &unallocated);
}

static void match_unwind_function(const char *fn, struct expression *expr, void
{
        set_state(my_id, "unwind_function", NULL, &called);
}

static int func_returns_int(void)
{
        struct symbol *type;

        type = get_base_type(cur_func_sym);
        if (!type || type->type != SYM_FN)
                return 0;
```

```
127         type = get_base_type(type);
128         if (type->ctype.base_type == &int_type) {
129                 return 1;
130         }
131         return 0;
132 }

134 static void match_return(struct expression *ret_value)
135 {
136         struct stree *stree;
137         struct sm_state *tmp;
138         sval_t sval;

140         if (!func_returns_int())
141                 return;
142         if (get_value(ret_value, &sval) && sval_cmp_val(sval, 0) >= 0)
143                 return;
144         if (!implied_not_equal(ret_value, 0))
145                 return;
146         if (get_state(my_id, "unwind_function", NULL) == &called)
147                 return;

149         stree = __get_cur_stree();
150         FOR_EACH_MY_SM(my_id, stree, tmp) {
151                 if (slist_has_state(tmp->possible, &allocated))
152                         sm_warning("'%s' was not released on error", tmp->name);
153         } END_FOR_EACH_SM(tmp);
154 }

156 static void register_unwind_functions(void)
157 {
158         struct token *token;
159         const char *func;

161         token = get_tokens_file("kernel.unwind_functions");
162         if (!token)
163                 return;
164         if (token_type(token) != TOKEN_STREAMBEGIN)
165                 return;
166         token = token->next;
167         while (token_type(token) != TOKEN_STREAMEND) {
168                 if (token_type(token) != TOKEN_IDENT)
169                         return;
170                 func = show_ident(token->ident);
171                 add_function_hook(func, &match_unwind_function, NULL);
172                 token = token->next;
173         }
174         clear_token_alloc();
175 }

177 static void release_function_indicator(const char *name)
178 {
179         if (!option_info)
180                 return;
181         add_function_hook(name, &print_unwind_functions, INT_PTR(0));
182 }

184 void check_unwind(int id)
185 {
186         if (option_project != PROJ_KERNEL || !option_spammy)
187                 return;
188         my_id = id;

190         register_unwind_functions();

192         return_implies_state("request_resource", 0, 0, &request_granted, INT_PTR
```



```
193         return_implies_state("request_resource", -EBUSY, -EBUSY, &request_denied
194         add_function_hook("release_resource", &match_release, INT_PTR(0));
195         release_function_indicator("release_resource");

197         return_implies_state("__request_region", valid_ptr_min, valid_ptr_max, &
198         return_implies_state("__request_region", 0, 0, &request_denied, INT_PTR(
199         add_function_hook("__release_region", &match_release, INT_PTR(1));
200         release_function_indicator("__release_region");

202         return_implies_state("ioremap", valid_ptr_min, valid_ptr_max, &request_g
203         return_implies_state("ioremap", 0, 0, &request_denied, INT_PTR(-1));
204         add_function_hook("iounmap", &match_release, INT_PTR(0));

206         return_implies_state("pci_iomap", valid_ptr_min, valid_ptr_max, &request
207         return_implies_state("pci_iomap", 0, 0, &request_denied, INT_PTR(-1));
208         add_function_hook("pci_iounmap", &match_release, INT_PTR(1));
209         release_function_indicator("pci_iounmap");

211         return_implies_state("__create_workqueue_key", valid_ptr_min, valid_ptr_
212                              INT_PTR(-1));
213         return_implies_state("__create_workqueue_key", 0, 0, &request_denied, IN
214         add_function_hook("destroy_workqueue", &match_release, INT_PTR(0));

216         return_implies_state("request_irq", 0, 0, &request_granted, INT_PTR(0));
217         return_implies_state("request_irq", -MAX_ERRNO, -1, &request_denied, INT
218         add_function_hook("free_irq", &match_release, INT_PTR(0));
219         release_function_indicator("free_irq");

221         return_implies_state("register_netdev", 0, 0, &request_granted, INT_PTR(
222         return_implies_state("register_netdev", -MAX_ERRNO, -1, &request_denied,
223         add_function_hook("unregister_netdev", &match_release, INT_PTR(0));
224         release_function_indicator("unregister_netdev");

226         return_implies_state("misc_register", 0, 0, &request_granted, INT_PTR(0)
227         return_implies_state("misc_register", -MAX_ERRNO, -1, &request_denied, I
228         add_function_hook("misc_deregister", &match_release, INT_PTR(0));
229         release_function_indicator("misc_deregister");


232         add_hook(&match_return, RETURN_HOOK);
233 }
```

```
**********************************************************
    1280 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_wait_for_common.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_wait_for_common(const char *fn, struct expression *expr, void
{
        char *name;

        if (!expr_unsigned(expr->left))
                return;
        name = expr_to_str(expr->left);
        sm_error("'%s()' returns negative and '%s' is unsigned", fn, name);
        free_string(name);
}

void check_wait_for_common(int id)
{
        my_id = id;

        if (option_project != PROJ_KERNEL)
                return;
        add_function_assign_hook("wait_for_common", &match_wait_for_common, NULL
        add_function_assign_hook("wait_for_completion_interruptible_timeout", &m
}
```

```
**********************************************************
    1280 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_wine.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2016 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is wine specific stuff for smatch_extra.
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_extra.h"

/* report (R_FATAL, "Can't get OS version."); */
void match_fatal_report(const char *fn, struct expression *expr,
                        void *unused)
{
        struct expression *arg;
        sval_t sval;

        arg = get_argument_from_call_expr(expr->args, 0);
        if (!get_implied_value(arg, &sval))
                return;

        /* R_FATAL is 9. */
        if (sval.value == 9)
                nullify_path();
}


void check_wine(int id)
{
        if (option_project != PROJ_WINE)
                return;

        add_function_hook("report", &match_fatal_report, NULL);
}
```

```
**********************************************************
    1976 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_wine_WtoA.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Idea from Michael Stefaniuc and Vincent Bˆ'ron's earlier WtoA
 * check.
 *
 * Apparently when you are coding WINE, you are not allowed to call
 * functions that end in capital 'A' from functions that end in
 * capital 'W'
 *
 */

#include "smatch.h"

static int my_id;

static int in_w = 0;

static void match_function_def(struct symbol *sym)
{
        char *func = get_function();
        int len;

        if (!func) {
                in_w = 0;
                return;
        }
        len = strlen(func);
        if (func[len - 1] == 'W' && len > 2 && func[len - 2] != 'A' )
                in_w = 1;
        else
                in_w = 0;
}

static int allowed_func(const char *fn)
{
        if (!strcmp("lstrcatA", fn))
                return 1;
        if (!strcmp("lstrcpyA", fn))
                return 1;
        if (!strcmp("lstrcpynA", fn))
                return 1;
        if (!strcmp("lstrlenA", fn))
                return 1;
        return 0;
```



```
}

static void match_call(struct expression *expr)
{
        char *fn_name;
        int len;

        if (!in_w)
                return;

        fn_name = expr_to_var(expr->fn);
        if (!fn_name)
                goto free;
        len = strlen(fn_name);
        if (fn_name[len - 1] == 'A' && !allowed_func(fn_name)) {
                sm_warning("WtoA call '%s()'", fn_name);
        }
free:
        free_string(fn_name);
}

void check_wine_WtoA(int id)
{
        if (option_project != PROJ_WINE)
                return;

        my_id = id;
        add_hook(&match_function_def, FUNC_DEF_HOOK);
        add_hook(&match_call, FUNCTION_CALL_HOOK);
}
```

```
**********************************************************
    2368 Fri Dec 21 15:00:09 2018
new/usr/src/tools/smatch/src/check_wine_filehandles.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * In wine you aren't allowed to compare file handles with 0,
 * only with INVALID_HANDLE_VALUE.
 *
 */

#include "smatch.h"

static int my_id;

STATE(filehandle);
STATE(oktocheck);

/* originally:
 *  "(?:CreateFile|CreateMailslot|CreateNamedPipe|FindFirstFile(?:Ex)?|OpenConso
 *
 */
static const char *filehandle_funcs[] = {
        "CreateFile",
        "CreateMailslot",
        "CreateNamedPipe",
        "FindFirstFile",
        "FindFirstFileEx",
        "OpenConsole",
        "SetupOpenInfFile",
        "socket",
        NULL,
};

static void ok_to_use(struct sm_state *sm, struct expression *mod_expr)
{
        if (sm->state != &oktocheck)
                set_state(my_id, sm->name, sm->sym, &oktocheck);
}

static void match_returns_handle(const char *fn, struct expression *expr,
                                 void *info)
{
        char *left_name = NULL;
        struct symbol *left_sym;

        left_name = expr_to_var_sym(expr->left, &left_sym);
        if (!left_name || !left_sym)
```



```
                goto free;
        set_state_expr(my_id, expr->left, &filehandle);
free:
        free_string(left_name);
}

static void match_condition(struct expression *expr)
{
        if (expr->type == EXPR_ASSIGNMENT)
                match_condition(expr->left);

        if (get_state_expr(my_id, expr) == &filehandle) {
                char *name;

                name = expr_to_var(expr);
                sm_error("comparing a filehandle against zero '%s'", name);
                set_state_expr(my_id, expr, &oktocheck);
                free_string(name);
        }
}

void check_wine_filehandles(int id)
{
        int i;

        if (option_project != PROJ_WINE)
                return;

        my_id = id;
        for (i = 0; filehandle_funcs[i]; i++) {
                add_function_assign_hook(filehandle_funcs[i],
                                         &match_returns_handle, NULL);
        }
        add_hook(&match_condition, CONDITION_HOOK);
        add_modification_hook(my_id, ok_to_use);
}
```

```
**********************************************************
    2394 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/check_wrong_size_arg.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static int my_id;

static void match_parameter(const char *fn, struct expression *expr, void *_para
{
        int param = PTR_INT(_param);
        struct expression *arg;
        char *name;

        arg = get_argument_from_call_expr(expr->args, param);
        arg = strip_expr(arg);
        if (!arg)
                return;
        if (arg->type != EXPR_COMPARE)
                return;

        name = expr_to_str_sym(arg, NULL);
        sm_warning("expected a buffer size but got a comparison '%s'", name);
        free_string(name);
}

static void register_funcs_from_file(void)
{
        char name[256];
        struct token *token;
        const char *func;
        char prev_func[256];
        int size;

        memset(prev_func, 0, sizeof(prev_func));
        snprintf(name, 256, "%s.sizeof_param", option_project_str);
        name[255] = '\0';
        token = get_tokens_file(name);
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        break;
                func = show_ident(token->ident);
```



```
                token = token->next;
                if (token_type(token) != TOKEN_NUMBER)
                        break;
                size = atoi(token->number);

                token = token->next;
                if (token_type(token) == TOKEN_SPECIAL) {
                        if (token->special != '-')
                                break;
                        token = token->next;
                }
                if (token_type(token) != TOKEN_NUMBER)
                        break;
                /* we don't care which argument hold the buf pointer */
                token = token->next;

                if (strcmp(func, prev_func) == 0)
                        continue;
                strncpy(prev_func, func, 255);

                add_function_hook(func, &match_parameter, INT_PTR(size));

        }
        if (token_type(token) != TOKEN_STREAMEND)
                sm_perror("problem parsing '%s'", name);
        clear_token_alloc();
}

void check_wrong_size_arg(int id)
{
        my_id = id;
        register_funcs_from_file();
}
```

```
**********************************************************
    3599 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/check_zero_to_err_ptr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

static int is_comparison_call(struct expression *expr)
{
        expr = expr_get_parent_expr(expr);
        if (!expr || expr->type != EXPR_COMPARE)
                return 0;
        if (expr->op != SPECIAL_EQUAL && expr->op != SPECIAL_NOTEQUAL)
                return 0;
        return 1;
}

static int next_line_is_if(struct expression *expr)
{
        struct expression *next;

        if (!__next_stmt || __next_stmt->type != STMT_IF)
                return 0;

        next = strip_expr(__next_stmt->if_conditional);
        while (next->type == EXPR_PREOP && next->op == '!')
                next = strip_expr(next->unop);
        if (expr_equiv(expr, next))
                return 1;
        return 0;
}

static int next_line_checks_IS_ERR(struct expression *call, struct expression *a
{
        struct expression *next;
        struct expression *tmp;

        tmp = expr_get_parent_expr(call);
        if (tmp && tmp->type == EXPR_ASSIGNMENT) {
                if (next_line_checks_IS_ERR(NULL, tmp->left))
                        return 1;
        }

        if (!__next_stmt || __next_stmt->type != STMT_IF)
```



```
                return 0;

        next = strip_expr(__next_stmt->if_conditional);
        while (next->type == EXPR_PREOP && next->op == '!')
                next = strip_expr(next->unop);
        if (!next || next->type != EXPR_CALL)
                return 0;
        if (next->fn->type != EXPR_SYMBOL || !next->fn->symbol ||
            !next->fn->symbol->ident ||
            (strcmp(next->fn->symbol->ident->name, "IS_ERR") != 0 &&
             strcmp(next->fn->symbol->ident->name, "IS_ERR_OR_NULL") != 0))
                return 0;
        next = get_argument_from_call_expr(next->args, 0);
        return expr_equiv(next, arg);
}

static int is_valid_ptr(sval_t sval)
{
        if (sval.type == &int_ctype &&
            (sval.value == INT_MIN || sval.value == INT_MAX))
                return 0;

        if (sval_cmp(valid_ptr_min_sval, sval) <= 0 &&
            sval_cmp(valid_ptr_max_sval, sval) >= 0)
                return 1;
        return 0;
}

static void match_err_ptr(const char *fn, struct expression *expr, void *data)
{
        struct expression *arg_expr;
        struct sm_state *sm, *tmp;
        sval_t sval;

        arg_expr = get_argument_from_call_expr(expr->args, 0);
        sm = get_sm_state_expr(SMATCH_EXTRA, arg_expr);
        if (!sm)
                return;

        if (is_comparison_call(expr))
                return;

        if (next_line_checks_IS_ERR(expr, arg_expr))
                return;
        if (strcmp(fn, "ERR_PTR") == 0 &&
            next_line_is_if(arg_expr))
                return;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (!estate_rl(tmp->state))
                        continue;
                if (is_valid_ptr(estate_min(tmp->state)) &&
                    is_valid_ptr(estate_max(tmp->state))) {
                        sm_warning("passing a valid pointer to '%s'", fn);
                        return;
                }
                if (!rl_to_sval(estate_rl(tmp->state), &sval))
                        continue;
                if (sval.value != 0)
                        continue;
                sm_warning("passing zero to '%s'", fn);
                return;
        } END_FOR_EACH_PTR(tmp);
}

void check_zero_to_err_ptr(int id)
```

```
127 {
128         if (option_project != PROJ_KERNEL)
129                 return;
130
131         my_id = id;
132         add_function_hook("ERR_PTR", &match_err_ptr, NULL);
133         add_function_hook("ERR_CAST", &match_err_ptr, NULL);
134         add_function_hook("PTR_ERR", &match_err_ptr, NULL);
135 }
```

**********
1343 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat-bsd.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
/*
 * BSD Compatibility functions
 *
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include <sys/types.h>
#include <string.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"

#include "compat/mmap-blob.c"

long double string_to_ld(const char *nptr, char **endptr)
{
        return strtod(nptr, endptr);
}
```

**********
*1747 Fri Dec 21 15:00:10 2018*
*new/usr/src/tools/smatch/src/compat-cygwin.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Cygwin Compatibility functions
 *
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */



#include <sys/mman.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"

void *blob_alloc(unsigned long size)
{
        void *ptr;
        size = (size + 4095) & ~4095;
        ptr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMO
        if (ptr == MAP_FAILED)
                ptr = NULL;
        else
                memset(ptr, 0, size);
        return ptr;
}

void blob_free(void *addr, unsigned long size)
{
        size = (size + 4095) & ~4095;
        munmap(addr, size);
}

long double string_to_ld(const char *nptr, char **endptr)
{
        return strtod(nptr, endptr);
}
```

```
**********************************************************
     119 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat-linux.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #define _GNU_SOURCE

   3 #include "lib.h"
   4 #include "allocate.h"

   6 #include "compat/mmap-blob.c"
   7 #include "compat/strtold.c"
```

```
**********************************************************
    1602 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat-mingw.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * MinGW Compatibility functions
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */



#include <stdarg.h>
#include <windef.h>
#include <winbase.h>
#include <stdlib.h>
#include <string.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"

void *blob_alloc(unsigned long size)
{
        void *ptr;
        ptr = malloc(size);
        if (ptr != NULL)
                memset(ptr, 0, size);
        return ptr;
}

void blob_free(void *addr, unsigned long size)
{
        free(addr);
}

long double string_to_ld(const char *nptr, char **endptr)
{
        return strtod(nptr, endptr);
}
```

```
**********************************************************
     685 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat-solaris.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "lib.h"
#include "allocate.h"

#include "compat/mmap-blob.c"

#include <floatingpoint.h>
#include <limits.h>
#include <errno.h>

long double string_to_ld(const char *str, char **endptr)
{
        long double res;
        decimal_record dr;
        enum decimal_string_form form;
        decimal_mode dm;
        fp_exception_field_type excp;
        char *echar;

        string_to_decimal ((char **)&str, INT_MAX, 0,
                           &dr, &form, &echar);
        if (endptr) *endptr = (char *)str;

        if (form == invalid_form) {
                errno = EINVAL;
                return 0.0;
        }

        dm.rd = fp_nearest;
        decimal_to_quadruple (&res, &dm, &dr, &excp);
        if (excp & ((1 << fp_overflow) | (1 << fp_underflow)))
                errno = ERANGE;
        return res;
}
```

```
**********************************************************
     713 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef COMPAT_H
#define COMPAT_H

/*
 * Various systems get these things wrong. So
 * we create a small compat library for them.
 *
 *  - zeroed anonymous mmap
 *      Missing in MinGW
 *  - "string to long double" (C99 strtold())
 *      Missing in Solaris and MinGW
 */
struct stream;
struct stat;

/*
 * Our "blob" allocator works on chunks that are multiples
 * of this size (the underlying allocator may be a mmap that
 * cannot handle smaller chunks, for example, so trying to
 * allocate blobs that aren't aligned is not going to work).
 */
#define CHUNK 32768

void *blob_alloc(unsigned long size);
void blob_free(void *addr, unsigned long size);
long double string_to_ld(const char *nptr, char **endptr);

#endif
```

```
**********************************************************
    1218 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat/bswap.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef _COMPAT_BSWAP_H_
#define _COMPAT_BSWAP_H_

#if defined(__GNUC__)
#if (__GNUC__ > 4) || ((__GNUC__ == 4) && (__GNUC_MINOR__ >= 8))
#define __HAS_BUILTIN_BSWAP16
#endif
#if (__GNUC__ > 4) || ((__GNUC__ == 4) && (__GNUC_MINOR__ >= 4))
#define __HAS_BUILTIN_BSWAP32
#define __HAS_BUILTIN_BSWAP64
#endif
#endif

#if defined(__clang__)
#if (__clang_major__ > 3) || ((__clang_major__ == 3) && (__clang_minor__ >= 2))
#define __HAS_BUILTIN_BSWAP16
#endif
#if (__clang_major__ > 3) || ((__clang_major__ == 3) && (__clang_minor__ >= 0))
#define __HAS_BUILTIN_BSWAP32
#define __HAS_BUILTIN_BSWAP64
#endif
#endif

#ifdef __HAS_BUILTIN_BSWAP16
#define bswap16(x)      __builtin_bswap16(x)
#else
#include <stdint.h>
static inline uint16_t bswap16(uint16_t x)
{
        return x << 8 | x >> 8;
}
#endif

#ifdef __HAS_BUILTIN_BSWAP32
#define bswap32(x)      __builtin_bswap32(x)
#else
#include <stdint.h>
static inline uint32_t bswap32(uint32_t x)
{
        return x >> 24 | (x >> 8 & 0xff00) | (x << 8 & 0xff0000) | x << 24;
}
#endif

#ifdef __HAS_BUILTIN_BSWAP64
#define bswap64(x)      __builtin_bswap64(x)
#else
#include <stdint.h>
static inline uint64_t bswap64(uint64_t x)
{
        return ((uint64_t)bswap32(x)) << 32 | bswap32(x >> 32);
}
#endif

#endif
```

```
**********************************************************
     853 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat/mmap-blob.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <sys/mman.h>
#include <sys/types.h>

/*
 * Allow old BSD naming too, it would be a pity to have to make a
 * separate file just for this.
 */
#ifndef MAP_ANONYMOUS
#define MAP_ANONYMOUS MAP_ANON
#endif

/*
 * Our blob allocator enforces the strict CHUNK size
 * requirement, as a portability check.
 */
void *blob_alloc(unsigned long size)
{
        void *ptr;

        if (size & ~CHUNK)
                die("internal error: bad allocation size (%lu bytes)", size);
        ptr = mmap(NULL, size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMO
        if (ptr == MAP_FAILED)
                ptr = NULL;
        return ptr;
}

void blob_free(void *addr, unsigned long size)
{
        if (!size || (size & ~CHUNK) || ((unsigned long) addr & 512))
                die("internal error: bad blob free (%lu bytes at %p)", size, add
#ifndef DEBUG
        munmap(addr, size);
#else
        mprotect(addr, size, PROT_NONE);
#endif
}
```

```
**********************************************************
     114 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compat/strtold.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #include <stdlib.h>

   3 long double string_to_ld(const char *nptr, char **endptr)
   4 {
   5         return strtold(nptr, endptr);
   6 }
```

```
**********************************************************
   56354 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compile-i386.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/compile-i386.c
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 * Copyright 2003 Jeff Garzik
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * x86 backend
 *
 * TODO list:
 * in general, any non-32bit SYM_BASETYPE is unlikely to work.
 * complex initializers
 * bitfields
 * global struct/union variables
 * addressing structures, and members of structures (as opposed to
 *     scalars) on the stack.  Requires smarter stack frame allocation.
 * labels / goto
 * any function argument that isn't 32 bits (or promoted to such)
 * inline asm
 * floating point
 *
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <assert.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "scope.h"
#include "expression.h"
#include "target.h"
#include "compile.h"
#include "bitmap.h"
#include "version.h"
```



```
struct textbuf {
        unsigned int    len;    /* does NOT include terminating null */
        char            *text;
        struct textbuf  *next;
        struct textbuf  *prev;
};

struct loop_stack {
        int             continue_lbl;
        int             loop_bottom_lbl;
        struct loop_stack *next;
};

struct atom;
struct storage;
DECLARE_PTR_LIST(str_list, struct atom);
DECLARE_PTR_LIST(atom_list, struct atom);
DECLARE_PTR_LIST(storage_list, struct storage);

struct function {
        int stack_size;
        int pseudo_nr;
        struct storage_list *pseudo_list;
        struct atom_list *atom_list;
        struct str_list *str_list;
        struct loop_stack *loop_stack;
        struct symbol **argv;
        unsigned int argc;
        int ret_target;
};

enum storage_type {
        STOR_PSEUDO,    /* variable stored on the stack */
        STOR_ARG,       /* function argument */
        STOR_SYM,       /* a symbol we can directly ref in the asm */
        STOR_REG,       /* scratch register */
        STOR_VALUE,     /* integer constant */
        STOR_LABEL,     /* label / jump target */
        STOR_LABELSYM,  /* label generated from symbol's pointer value */
};

struct reg_info {
        const char      *name;
        struct storage  *contains;
        const unsigned char aliases[12];
#define own_regno aliases[0]
};

struct storage {
        enum storage_type type;
        unsigned long flags;

        /* STOR_REG */
        struct reg_info *reg;
        struct symbol *ctype;

        union {
                /* STOR_PSEUDO */
                struct {
                        int pseudo;
                        int offset;
                        int size;
                };
                /* STOR_ARG */
                struct {
```

```
127                         int idx;
128                 };
129                 /* STOR_SYM */
130                 struct {
131                         struct symbol *sym;
132                 };
133                 /* STOR_VALUE */
134                 struct {
135                         long long value;
136                 };
137                 /* STOR_LABEL */
138                 struct {
139                         int label;
140                 };
141                 /* STOR_LABELSYM */
142                 struct {
143                         struct symbol *labelsym;
144                 };
145         };
146 };

148 enum {
149         STOR_LABEL_VAL  = (1 << 0),
150         STOR_WANTS_FREE = (1 << 1),
151 };

153 struct symbol_private {
154         struct storage *addr;
155 };

157 enum atom_type {
158         ATOM_TEXT,
159         ATOM_INSN,
160         ATOM_CSTR,
161 };

163 struct atom {
164         enum atom_type type;
165         union {
166                 /* stuff for text */
167                 struct {
168                         char *text;
169                         unsigned int text_len;  /* w/o terminating null */
170                 };

172                 /* stuff for insns */
173                 struct {
174                         char insn[32];
175                         char comment[40];
176                         struct storage *op1;
177                         struct storage *op2;
178                 };

180                 /* stuff for C strings */
181                 struct {
182                         struct string *string;
183                         int label;
184                 };
185         };
186 };


189 static struct function *current_func = NULL;
190 static struct textbuf *unit_post_text = NULL;
191 static const char *current_section;
```



```
193 static void emit_comment(const char * fmt, ...) FORMAT_ATTR(1);
194 static void emit_move(struct storage *src, struct storage *dest,
195                       struct symbol *ctype, const char *comment);
196 static int type_is_signed(struct symbol *sym);
197 static struct storage *x86_address_gen(struct expression *expr);
198 static struct storage *x86_symbol_expr(struct symbol *sym);
199 static void x86_symbol(struct symbol *sym);
200 static struct storage *x86_statement(struct statement *stmt);
201 static struct storage *x86_expression(struct expression *expr);

203 enum registers {
204         NOREG,
205          AL,  DL,  CL,  BL,  AH,  DH,  CH,  BH, // 8-bit
206          AX,  DX,  CX,  BX,  SI,  DI,  BP,  SP, // 16-bit
207         EAX, EDX, ECX, EBX, ESI, EDI, EBP, ESP, // 32-bit
208         EAX_EDX, ECX_EBX, ESI_EDI,              // 64-bit
209 };

211 /* This works on regno's, reg_info's and hardreg_storage's */
212 #define byte_reg(reg) ((reg) - 16)
213 #define highbyte_reg(reg) ((reg)-12)
214 #define word_reg(reg) ((reg)-8)

216 #define REGINFO(nr, str, conflicts...)  [nr] = { .name = str, .aliases = { nr ,

218 static struct reg_info reg_info_table[] = {
219         REGINFO( AL,  "%al", AX, EAX, EAX_EDX),
220         REGINFO( DL,  "%dl", DX, EDX, EAX_EDX),
221         REGINFO( CL,  "%cl", CX, ECX, ECX_EBX),
222         REGINFO( BL,  "%bl", BX, EBX, ECX_EBX),
223         REGINFO( AH,  "%ah", AX, EAX, EAX_EDX),
224         REGINFO( DH,  "%dh", DX, EDX, EAX_EDX),
225         REGINFO( CH,  "%ch", CX, ECX, ECX_EBX),
226         REGINFO( BH,  "%bh", BX, EBX, ECX_EBX),
227         REGINFO( AX,  "%ax", AL, AH, EAX, EAX_EDX),
228         REGINFO( DX,  "%dx", DL, DH, EDX, EAX_EDX),
229         REGINFO( CX,  "%cx", CL, CH, ECX, ECX_EBX),
230         REGINFO( BX,  "%bx", BL, BH, EBX, ECX_EBX),
231         REGINFO( SI,  "%si", ESI, ESI_EDI),
232         REGINFO( DI,  "%di", EDI, ESI_EDI),
233         REGINFO( BP,  "%bp", EBP),
234         REGINFO( SP,  "%sp", ESP),
235         REGINFO(EAX, "%eax", AL, AH, AX, EAX_EDX),
236         REGINFO(EDX, "%edx", DL, DH, DX, EAX_EDX),
237         REGINFO(ECX, "%ecx", CL, CH, CX, ECX_EBX),
238         REGINFO(EBX, "%ebx", BL, BH, BX, ECX_EBX),
239         REGINFO(ESI, "%esi", SI, ESI_EDI),
240         REGINFO(EDI, "%edi", DI, ESI_EDI),
241         REGINFO(EBP, "%ebp", BP),
242         REGINFO(ESP, "%esp", SP),
243         REGINFO(EAX_EDX, "%eax:%edx", AL, AH, AX, EAX, DL, DH, DX, EDX),
244         REGINFO(ECX_EBX, "%ecx:%ebx", CL, CH, CX, ECX, BL, BH, BX, EBX),
245         REGINFO(ESI_EDI, "%esi:%edi", SI, ESI, DI, EDI),
246 };

248 #define REGSTORAGE(nr) [nr] = { .type = STOR_REG, .reg = reg_info_table + (nr) }

250 static struct storage hardreg_storage_table[] = {
251         REGSTORAGE(AL), REGSTORAGE(DL), REGSTORAGE(CL), REGSTORAGE(BL),
252         REGSTORAGE(AH), REGSTORAGE(DH), REGSTORAGE(CH), REGSTORAGE(BH),
253         REGSTORAGE(AX), REGSTORAGE(DX), REGSTORAGE(CX), REGSTORAGE(BX),
254         REGSTORAGE(SI), REGSTORAGE(DI), REGSTORAGE(BP), REGSTORAGE(SP),
255         REGSTORAGE(EAX), REGSTORAGE(EDX), REGSTORAGE(ECX), REGSTORAGE(EBX),
256         REGSTORAGE(ESI), REGSTORAGE(EDI), REGSTORAGE(EBP), REGSTORAGE(ESP),
257         REGSTORAGE(EAX_EDX), REGSTORAGE(ECX_EBX), REGSTORAGE(ESI_EDI),
258 };
```

```
#define REG_EAX (&hardreg_storage_table[EAX])
#define REG_ECX (&hardreg_storage_table[ECX])
#define REG_EDX (&hardreg_storage_table[EDX])
#define REG_ESP (&hardreg_storage_table[ESP])
#define REG_DL  (&hardreg_storage_table[DL])
#define REG_DX  (&hardreg_storage_table[DX])
#define REG_AL  (&hardreg_storage_table[AL])
#define REG_AX  (&hardreg_storage_table[AX])

static DECLARE_BITMAP(regs_in_use, 256);

static inline struct storage * reginfo_reg(struct reg_info *info)
{
        return hardreg_storage_table + info->own_regno;
}

static struct storage * get_hardreg(struct storage *reg, int clear)
{
        struct reg_info *info = reg->reg;
        const unsigned char *aliases;
        int regno;

        aliases = info->aliases;
        while ((regno = *aliases++) != NOREG) {
                if (test_bit(regno, regs_in_use))
                        goto busy;
                if (clear)
                        reg_info_table[regno].contains = NULL;
        }
        set_bit(info->own_regno, regs_in_use);
        return reg;
busy:
        fprintf(stderr, "register %s is busy\n", info->name);
        if (regno + reg_info_table != info)
                fprintf(stderr, "  conflicts with %s\n", reg_info_table[regno].n
        exit(1);
}

static void put_reg(struct storage *reg)
{
        struct reg_info *info = reg->reg;
        int regno = info->own_regno;

        if (test_and_clear_bit(regno, regs_in_use))
                return;
        fprintf(stderr, "freeing already free'd register %s\n", reg_info_table[r
}

struct regclass {
        const char *name;
        const unsigned char regs[30];
};

static struct regclass regclass_8 = { "8-bit", { AL, DL, CL, BL, AH, DH, CH, BH
static struct regclass regclass_16 = { "16-bit", { AX, DX, CX, BX, SI, DI, BP }}
static struct regclass regclass_32 = { "32-bit", { EAX, EDX, ECX, EBX, ESI, EDI,
static struct regclass regclass_64 = { "64-bit", { EAX_EDX, ECX_EBX, ESI_EDI }};

static struct regclass regclass_32_8 = { "32-bit bytes", { EAX, EDX, ECX, EBX }}

static struct regclass *get_regclass_bits(int bits)
{
        switch (bits) {
        case 8: return &regclass_8;
        case 16: return &regclass_16;
```



```
        case 64: return &regclass_64;
        default: return &regclass_32;
        }
}

static struct regclass *get_regclass(struct expression *expr)
{
        return get_regclass_bits(expr->ctype->bit_size);
}

static int register_busy(int regno)
{
        if (!test_bit(regno, regs_in_use)) {
                struct reg_info *info = reg_info_table + regno;
                const unsigned char *regs = info->aliases+1;

                while ((regno = *regs) != NOREG) {
                        regs++;
                        if (test_bit(regno, regs_in_use))
                                goto busy;
                }
                return 0;
        }
busy:
        return 1;
}

static struct storage *get_reg(struct regclass *class)
{
        const unsigned char *regs = class->regs;
        int regno;

        while ((regno = *regs) != NOREG) {
                regs++;
                if (register_busy(regno))
                        continue;
                return get_hardreg(hardreg_storage_table + regno, 1);
        }
        fprintf(stderr, "Ran out of %s registers\n", class->name);
        exit(1);
}

static struct storage *get_reg_value(struct storage *value, struct regclass *cla
{
        struct reg_info *info;
        struct storage *reg;

        /* Do we already have it somewhere */
        info = value->reg;
        if (info && info->contains == value) {
                emit_comment("already have register %s", info->name);
                return get_hardreg(hardreg_storage_table + info->own_regno, 0);
        }

        reg = get_reg(class);
        emit_move(value, reg, value->ctype, "reload register");
        info = reg->reg;
        info->contains = value;
        value->reg = info;
        return reg;
}

static struct storage *temp_from_bits(unsigned int bit_size)
{
        return get_reg(get_regclass_bits(bit_size));
}
```

```
392 static inline unsigned int pseudo_offset(struct storage *s)
393 {
394         if (s->type != STOR_PSEUDO)
395                 return 123456;  /* intentionally bogus value */
396
397         return s->offset;
398 }
399
400 static inline unsigned int arg_offset(struct storage *s)
401 {
402         if (s->type != STOR_ARG)
403                 return 123456;  /* intentionally bogus value */
404
405         /* FIXME: this is wrong wrong wrong */
406         return current_func->stack_size + ((1 + s->idx) * 4);
407 }
408
409 static const char *pretty_offset(int ofs)
410 {
411         static char esp_buf[64];
412
413         if (ofs)
414                 sprintf(esp_buf, "%d(%%esp)", ofs);
415         else
416                 strcpy(esp_buf, "(%esp)");
417
418         return esp_buf;
419 }
420
421 static void stor_sym_init(struct symbol *sym)
422 {
423         struct storage *stor;
424         struct symbol_private *priv;
425
426         priv = calloc(1, sizeof(*priv) + sizeof(*stor));
427         if (!priv)
428                 die("OOM in stor_sym_init");
429
430         stor = (struct storage *) (priv + 1);
431
432         priv->addr = stor;
433         stor->type = STOR_SYM;
434         stor->sym = sym;
435 }
436
437 static const char *stor_op_name(struct storage *s)
438 {
439         static char name[32];
440
441         switch (s->type) {
442         case STOR_PSEUDO:
443                 strcpy(name, pretty_offset((int) pseudo_offset(s)));
444                 break;
445         case STOR_ARG:
446                 strcpy(name, pretty_offset((int) arg_offset(s)));
447                 break;
448         case STOR_SYM:
449                 strcpy(name, show_ident(s->sym->ident));
450                 break;
451         case STOR_REG:
452                 strcpy(name, s->reg->name);
453                 break;
454         case STOR_VALUE:
455                 sprintf(name, "$%Ld", s->value);
456                 break;
```



```
457         case STOR_LABEL:
458                 sprintf(name, "%s.L%d", s->flags & STOR_LABEL_VAL ? "$" : "",
459                         s->label);
460                 break;
461         case STOR_LABELSYM:
462                 sprintf(name, "%s.LS%p", s->flags & STOR_LABEL_VAL ? "$" : "",
463                         s->labelsym);
464                 break;
465         }
466
467         return name;
468 }
469
470 static struct atom *new_atom(enum atom_type type)
471 {
472         struct atom *atom;
473
474         atom = calloc(1, sizeof(*atom));        /* TODO: chunked alloc */
475         if (!atom)
476                 die("nuclear OOM");
477
478         atom->type = type;
479
480         return atom;
481 }
482
483 static inline void push_cstring(struct function *f, struct string *str,
484                                 int label)
485 {
486         struct atom *atom;
487
488         atom = new_atom(ATOM_CSTR);
489         atom->string = str;
490         atom->label = label;
491
492         add_ptr_list(&f->str_list, atom);       /* note: _not_ atom_list */
493 }
494
495 static inline void push_atom(struct function *f, struct atom *atom)
496 {
497         add_ptr_list(&f->atom_list, atom);
498 }
499
500 static void push_text_atom(struct function *f, const char *text)
501 {
502         struct atom *atom = new_atom(ATOM_TEXT);
503
504         atom->text = strdup(text);
505         atom->text_len = strlen(text);
506
507         push_atom(f, atom);
508 }
509
510 static struct storage *new_storage(enum storage_type type)
511 {
512         struct storage *stor;
513
514         stor = calloc(1, sizeof(*stor));
515         if (!stor)
516                 die("OOM in new_storage");
517
518         stor->type = type;
519
520         return stor;
521 }
```

```
523 static struct storage *stack_alloc(int n_bytes)
524 {
525         struct function *f = current_func;
526         struct storage *stor;

528         assert(f != NULL);

530         stor = new_storage(STOR_PSEUDO);
531         stor->type = STOR_PSEUDO;
532         stor->pseudo = f->pseudo_nr;
533         stor->offset = f->stack_size; /* FIXME: stack req. natural align */
534         stor->size = n_bytes;
535         f->stack_size += n_bytes;
536         f->pseudo_nr++;

538         add_ptr_list(&f->pseudo_list, stor);

540         return stor;
541 }

543 static struct storage *new_labelsym(struct symbol *sym)
544 {
545         struct storage *stor;

547         stor = new_storage(STOR_LABELSYM);

549         if (stor) {
550                 stor->flags |= STOR_WANTS_FREE;
551                 stor->labelsym = sym;
552         }

554         return stor;
555 }

557 static struct storage *new_val(long long value)
558 {
559         struct storage *stor;

561         stor = new_storage(STOR_VALUE);

563         if (stor) {
564                 stor->flags |= STOR_WANTS_FREE;
565                 stor->value = value;
566         }

568         return stor;
569 }

571 static int new_label(void)
572 {
573         static int label = 0;
574         return ++label;
575 }

577 static void textbuf_push(struct textbuf **buf_p, const char *text)
578 {
579         struct textbuf *tmp, *list = *buf_p;
580         unsigned int text_len = strlen(text);
581         unsigned int alloc_len = text_len + 1 + sizeof(*list);

583         tmp = calloc(1, alloc_len);
584         if (!tmp)
585                 die("OOM on textbuf alloc");

587         tmp->text = ((void *) tmp) + sizeof(*tmp);
588         memcpy(tmp->text, text, text_len + 1);
```



```
589         tmp->len = text_len;

591         /* add to end of list */
592         if (!list) {
593                 list = tmp;
594                 tmp->prev = tmp;
595         } else {
596                 tmp->prev = list->prev;
597                 tmp->prev->next = tmp;
598                 list->prev = tmp;
599         }
600         tmp->next = list;

602         *buf_p = list;
603 }

605 static void textbuf_emit(struct textbuf **buf_p)
606 {
607         struct textbuf *tmp, *list = *buf_p;

609         while (list) {
610                 tmp = list;
611                 if (tmp->next == tmp)
612                         list = NULL;
613                 else {
614                         tmp->prev->next = tmp->next;
615                         tmp->next->prev = tmp->prev;
616                         list = tmp->next;
617                 }

619                 fputs(tmp->text, stdout);

621                 free(tmp);
622         }

624         *buf_p = list;
625 }

627 static void insn(const char *insn, struct storage *op1, struct storage *op2,
628                  const char *comment_in)
629 {
630         struct function *f = current_func;
631         struct atom *atom = new_atom(ATOM_INSN);

633         assert(insn != NULL);

635         strcpy(atom->insn, insn);
636         if (comment_in && (*comment_in))
637                 strncpy(atom->comment, comment_in,
638                         sizeof(atom->comment) - 1);

640         atom->op1 = op1;
641         atom->op2 = op2;

643         push_atom(f, atom);
644 }

646 static void emit_comment(const char *fmt, ...)
647 {
648         struct function *f = current_func;
649         static char tmpbuf[100] = "\t# ";
650         va_list args;
651         int i;

653         va_start(args, fmt);
654         i = vsnprintf(tmpbuf+3, sizeof(tmpbuf)-4, fmt, args);
```

```
655         va_end(args);
656         tmpbuf[i+3] = '\n';
657         tmpbuf[i+4] = '\0';
658         push_text_atom(f, tmpbuf);
659 }

661 static void emit_label (int label, const char *comment)
662 {
663         struct function *f = current_func;
664         char s[64];

666         if (!comment)
667                 sprintf(s, ".L%d:\n", label);
668         else
669                 sprintf(s, ".L%d:\t\t\t\t\t# %s\n", label, comment);

671         push_text_atom(f, s);
672 }

674 static void emit_labelsym (struct symbol *sym, const char *comment)
675 {
676         struct function *f = current_func;
677         char s[64];

679         if (!comment)
680                 sprintf(s, ".LS%p:\n", sym);
681         else
682                 sprintf(s, ".LS%p:\t\t\t\t# %s\n", sym, comment);

684         push_text_atom(f, s);
685 }

687 void emit_unit_begin(const char *basename)
688 {
689         printf("\t.file\t\"%s\"\n", basename);
690 }

692 void emit_unit_end(void)
693 {
694         textbuf_emit(&unit_post_text);
695         printf("\t.ident\t\"sparse silly x86 backend (version %s)\"\n", SPARSE_V
696 }

698 /* conditionally switch sections */
699 static void emit_section(const char *s)
700 {
701         if (s == current_section)
702                 return;
703         if (current_section && (!strcmp(s, current_section)))
704                 return;

706         printf("\t%s\n", s);
707         current_section = s;
708 }

710 static void emit_insn_atom(struct function *f, struct atom *atom)
711 {
712         char s[128];
713         char comment[64];
714         struct storage *op1 = atom->op1;
715         struct storage *op2 = atom->op2;

717         if (atom->comment[0])
718                 sprintf(comment, "\t\t# %s", atom->comment);
719         else
720                 comment[0] = 0;
```



```
722         if (atom->op2) {
723                 char tmp[16];
724                 strcpy(tmp, stor_op_name(op1));
725                 sprintf(s, "\t%s\t%s, %s%s\n",
726                         atom->insn, tmp, stor_op_name(op2), comment);
727         } else if (atom->op1)
728                 sprintf(s, "\t%s\t%s%s%s\n",
729                         atom->insn, stor_op_name(op1),
730                         comment[0] ? "\t" : "", comment);
731         else
732                 sprintf(s, "\t%s\t%s%s\n",
733                         atom->insn,
734                         comment[0] ? "\t\t" : "", comment);

736         if (write(STDOUT_FILENO, s, strlen(s)) < 0)
737                 die("can't write to stdout");
738 }

740 static void emit_atom_list(struct function *f)
741 {
742         struct atom *atom;

744         FOR_EACH_PTR(f->atom_list, atom) {
745                 switch (atom->type) {
746                 case ATOM_TEXT: {
747                         if (write(STDOUT_FILENO, atom->text, atom->text_len) < 0
748                                 die("can't write to stdout");
749                         break;
750                 }
751                 case ATOM_INSN:
752                         emit_insn_atom(f, atom);
753                         break;
754                 case ATOM_CSTR:
755                         assert(0);
756                         break;
757                 }
758         } END_FOR_EACH_PTR(atom);
759 }

761 static void emit_string_list(struct function *f)
762 {
763         struct atom *atom;

765         emit_section(".section\t.rodata");

767         FOR_EACH_PTR(f->str_list, atom) {
768                 /* FIXME: escape " in string */
769                 printf(".L%d:\n", atom->label);
770                 printf("\t.string\t%s\n", show_string(atom->string));

772                 free(atom);
773         } END_FOR_EACH_PTR(atom);
774 }

776 static void func_cleanup(struct function *f)
777 {
778         struct storage *stor;
779         struct atom *atom;

781         FOR_EACH_PTR(f->atom_list, atom) {
782                 if ((atom->type == ATOM_TEXT) && (atom->text))
783                         free(atom->text);
784                 if (atom->op1 && (atom->op1->flags & STOR_WANTS_FREE))
785                         free(atom->op1);
786                 if (atom->op2 && (atom->op2->flags & STOR_WANTS_FREE))
```

```
787                         free(atom->op2);
788                 free(atom);
789         } END_FOR_EACH_PTR(atom);

791         FOR_EACH_PTR(f->pseudo_list, stor) {
792                 free(stor);
793         } END_FOR_EACH_PTR(stor);

795         free_ptr_list(&f->pseudo_list);
796         free(f);
797 }

799 /* function prologue */
800 static void emit_func_pre(struct symbol *sym)
801 {
802         struct function *f;
803         struct symbol *arg;
804         unsigned int i, argc = 0, alloc_len;
805         unsigned char *mem;
806         struct symbol_private *privbase;
807         struct storage *storage_base;
808         struct symbol *base_type = sym->ctype.base_type;

810         FOR_EACH_PTR(base_type->arguments, arg) {
811                 argc++;
812         } END_FOR_EACH_PTR(arg);

814         alloc_len =
815                 sizeof(*f) +
816                 (argc * sizeof(struct symbol *)) +
817                 (argc * sizeof(struct symbol_private)) +
818                 (argc * sizeof(struct storage));
819         mem = calloc(1, alloc_len);
820         if (!mem)
821                 die("OOM on func info");

823         f               =  (struct function *) mem;
824         mem             += sizeof(*f);
825         f->argv         =  (struct symbol **) mem;
826         mem             += (argc * sizeof(struct symbol *));
827         privbase        =  (struct symbol_private *) mem;
828         mem             += (argc * sizeof(struct symbol_private));
829         storage_base    =  (struct storage *) mem;

831         f->argc = argc;
832         f->ret_target = new_label();

834         i = 0;
835         FOR_EACH_PTR(base_type->arguments, arg) {
836                 f->argv[i] = arg;
837                 arg->aux = &privbase[i];
838                 storage_base[i].type = STOR_ARG;
839                 storage_base[i].idx = i;
840                 privbase[i].addr = &storage_base[i];
841                 i++;
842         } END_FOR_EACH_PTR(arg);

844         assert(current_func == NULL);
845         current_func = f;
846 }

848 /* function epilogue */
849 static void emit_func_post(struct symbol *sym)
850 {
851         const char *name = show_ident(sym->ident);
852         struct function *f = current_func;
```



```
853         int stack_size = f->stack_size;

855         if (f->str_list)
856                 emit_string_list(f);

858         /* function prologue */
859         emit_section(".text");
860         if ((sym->ctype.modifiers & MOD_STATIC) == 0)
861                 printf(".globl %s\n", name);
862         printf("\t.type\t%s, @function\n", name);
863         printf("%s:\n", name);

865         if (stack_size) {
866                 char pseudo_const[16];

868                 sprintf(pseudo_const, "$%d", stack_size);
869                 printf("\tsubl\t%s, %%esp\n", pseudo_const);
870         }

872         /* function epilogue */

874         /* jump target for 'return' statements */
875         emit_label(f->ret_target, NULL);

877         if (stack_size) {
878                 struct storage *val;

880                 val = new_storage(STOR_VALUE);
881                 val->value = (long long) (stack_size);
882                 val->flags = STOR_WANTS_FREE;

884                 insn("addl", val, REG_ESP, NULL);
885         }

887         insn("ret", NULL, NULL, NULL);

889         /* output everything to stdout */
890         fflush(stdout);         /* paranoia; needed? */
891         emit_atom_list(f);

893         /* function footer */
894         name = show_ident(sym->ident);
895         printf("\t.size\t%s, .-%s\n", name, name);

897         func_cleanup(f);
898         current_func = NULL;
899 }

901 /* emit object (a.k.a. variable, a.k.a. data) prologue */
902 static void emit_object_pre(const char *name, unsigned long modifiers,
903                             unsigned long alignment, unsigned int byte_size)
904 {
905         if ((modifiers & MOD_STATIC) == 0)
906                 printf(".globl %s\n", name);
907         emit_section(".data");
908         if (alignment)
909                 printf("\t.align %lu\n", alignment);
910         printf("\t.type\t%s, @object\n", name);
911         printf("\t.size\t%s, %d\n", name, byte_size);
912         printf("%s:\n", name);
913 }

915 /* emit value (only) for an initializer scalar */
916 static void emit_scalar(struct expression *expr, unsigned int bit_size)
917 {
918         const char *type;
```

```
919         long long ll;

921         assert(expr->type == EXPR_VALUE);

923         if (expr->value == 0ULL) {
924                 printf("\t.zero\t%d\n", bit_size / 8);
925                 return;
926         }

928         ll = (long long) expr->value;

930         switch (bit_size) {
931         case 8:         type = "byte";  ll = (char) ll; break;
932         case 16:        type = "value"; ll = (short) ll; break;
933         case 32:        type = "long";  ll = (int) ll; break;
934         case 64:        type = "quad";  break;
935         default:        type = NULL;    break;
936         }

938         assert(type != NULL);

940         printf("\t.%s\t%Ld\n", type, ll);
941 }

943 static void emit_global_noinit(const char *name, unsigned long modifiers,
944                                unsigned long alignment, unsigned int byte_size)
945 {
946         char s[64];

948         if (modifiers & MOD_STATIC) {
949                 sprintf(s, "\t.local\t%s\n", name);
950                 textbuf_push(&unit_post_text, s);
951         }
952         if (alignment)
953                 sprintf(s, "\t.comm\t%s,%d,%lu\n", name, byte_size, alignment);
954         else
955                 sprintf(s, "\t.comm\t%s,%d\n", name, byte_size);
956         textbuf_push(&unit_post_text, s);
957 }

959 static int ea_current, ea_last;

961 static void emit_initializer(struct symbol *sym,
962                              struct expression *expr)
963 {
964         int distance = ea_current - ea_last - 1;

966         if (distance > 0)
967                 printf("\t.zero\t%d\n", (sym->bit_size / 8) * distance);

969         if (expr->type == EXPR_VALUE) {
970                 struct symbol *base_type = sym->ctype.base_type;
971                 assert(base_type != NULL);

973                 emit_scalar(expr, sym->bit_size / get_expression_value(base_type
974                 return;
975         }
976         if (expr->type != EXPR_INITIALIZER)
977                 return;

979         assert(0); /* FIXME */
980 }

982 static int sort_array_cmp(const struct expression *a,
983                           const struct expression *b)
984 {
```



```
985         int a_ofs = 0, b_ofs = 0;

987         if (a->type == EXPR_POS)
988                 a_ofs = (int) a->init_offset;
989         if (b->type == EXPR_POS)
990                 b_ofs = (int) b->init_offset;

992         return a_ofs - b_ofs;
993 }

995 /* move to front-end? */
996 static void sort_array(struct expression *expr)
997 {
998         struct expression *entry, **list;
999         unsigned int elem, sorted, i;

1001        elem = expression_list_size(expr->expr_list);
1002        if (!elem)
1003                return;

1005        list = malloc(sizeof(entry) * elem);
1006        if (!list)
1007                die("OOM in sort_array");

1009        /* this code is no doubt evil and ignores EXPR_INDEX possibly
1010         * to its detriment and other nasty things.  improvements
1011         * welcome.
1012         */
1013        i = 0;
1014        sorted = 0;
1015        FOR_EACH_PTR(expr->expr_list, entry) {
1016                if ((entry->type == EXPR_POS) || (entry->type == EXPR_VALUE)) {
1017                        /* add entry to list[], in sorted order */
1018                        if (sorted == 0) {
1019                                list[0] = entry;
1020                                sorted = 1;
1021                        } else {
1022                                for (i = 0; i < sorted; i++)
1023                                        if (sort_array_cmp(entry, list[i]) <= 0)
1024                                                break;

1026                                /* If inserting into the middle of list[]
1027                                 * instead of appending, we memmove.
1028                                 * This is ugly, but thankfully
1029                                 * uncommon.  Input data with tons of
1030                                 * entries very rarely have explicit
1031                                 * offsets.  convert to qsort eventually...
1032                                 */
1033                                if (i != sorted)
1034                                        memmove(&list[i + 1], &list[i],
1035                                                (sorted - i) * sizeof(entry));
1036                                list[i] = entry;
1037                                sorted++;
1038                        }
1039                }
1040        } END_FOR_EACH_PTR(entry);

1042        i = 0;
1043        FOR_EACH_PTR(expr->expr_list, entry) {
1044                if ((entry->type == EXPR_POS) || (entry->type == EXPR_VALUE))
1045                        *THIS_ADDRESS(entry) = list[i++];
1046        } END_FOR_EACH_PTR(entry);

1048        free(list);
1049 }
```

```
1051 static void emit_array(struct symbol *sym)
1052 {
1053         struct symbol *base_type = sym->ctype.base_type;
1054         struct expression *expr = sym->initializer;
1055         struct expression *entry;

1057         assert(base_type != NULL);

1059         stor_sym_init(sym);

1061         ea_last = -1;

1063         emit_object_pre(show_ident(sym->ident), sym->ctype.modifiers,
1064                         sym->ctype.alignment,
1065                         sym->bit_size / 8);

1067         sort_array(expr);

1069         FOR_EACH_PTR(expr->expr_list, entry) {
1070                 if (entry->type == EXPR_VALUE) {
1071                         ea_current = 0;
1072                         emit_initializer(sym, entry);
1073                         ea_last = ea_current;
1074                 } else if (entry->type == EXPR_POS) {
1075                         ea_current =
1076                             entry->init_offset / (base_type->bit_size / 8);
1077                         emit_initializer(sym, entry->init_expr);
1078                         ea_last = ea_current;
1079                 }
1080         } END_FOR_EACH_PTR(entry);
1081 }

1083 void emit_one_symbol(struct symbol *sym)
1084 {
1085         x86_symbol(sym);
1086 }

1088 static void emit_copy(struct storage *dest, struct storage *src,
1089                       struct symbol *ctype)
1090 {
1091         struct storage *reg = NULL;
1092         unsigned int bit_size;

1094         /* FIXME: Bitfield copy! */

1096         bit_size = src->size * 8;
1097         if (!bit_size)
1098                 bit_size = 32;
1099         if ((src->type == STOR_ARG) && (bit_size < 32))
1100                 bit_size = 32;

1102         reg = temp_from_bits(bit_size);
1103         emit_move(src, reg, ctype, "begin copy ..");

1105         bit_size = dest->size * 8;
1106         if (!bit_size)
1107                 bit_size = 32;
1108         if ((dest->type == STOR_ARG) && (bit_size < 32))
1109                 bit_size = 32;

1111         emit_move(reg, dest, ctype, ".... end copy");
1112         put_reg(reg);
1113 }

1115 static void emit_store(struct expression *dest_expr, struct storage *dest,
1116                        struct storage *src, int bits)
```



```
1117 {
1118         /* FIXME: Bitfield store! */
1119         printf("\tst.%d\t\tv%d,[v%d]\n", bits, src->pseudo, dest->pseudo);
1120 }

1122 static void emit_scalar_noinit(struct symbol *sym)
1123 {
1124         emit_global_noinit(show_ident(sym->ident),
1125                            sym->ctype.modifiers, sym->ctype.alignment,
1126                            sym->bit_size / 8);
1127         stor_sym_init(sym);
1128 }

1130 static void emit_array_noinit(struct symbol *sym)
1131 {
1132         emit_global_noinit(show_ident(sym->ident),
1133                            sym->ctype.modifiers, sym->ctype.alignment,
1134                            get_expression_value(sym->array_size) * (sym->bit_siz
1135         stor_sym_init(sym);
1136 }

1138 static const char *opbits(const char *insn, unsigned int bits)
1139 {
1140         static char opbits_str[32];
1141         char c;

1143         switch (bits) {
1144         case 8:  c = 'b'; break;
1145         case 16: c = 'w'; break;
1146         case 32: c = 'l'; break;
1147         case 64: c = 'q'; break;
1148         default: abort(); break;
1149         }

1151         sprintf(opbits_str, "%s%c", insn, c);

1153         return opbits_str;
1154 }

1156 static void emit_move(struct storage *src, struct storage *dest,
1157                       struct symbol *ctype, const char *comment)
1158 {
1159         unsigned int bits;
1160         unsigned int is_signed;
1161         unsigned int is_dest = (src->type == STOR_REG);
1162         const char *opname;

1164         if (ctype) {
1165                 bits = ctype->bit_size;
1166                 is_signed = type_is_signed(ctype);
1167         } else {
1168                 bits = 32;
1169                 is_signed = 0;
1170         }

1172         /*
1173          * Are we moving from a register to a register?
1174          * Make the new reg to be the "cache".
1175          */
1176         if ((dest->type == STOR_REG) && (src->type == STOR_REG)) {
1177                 struct storage *backing;

1179 reg_reg_move:
1180                 if (dest == src)
1181                         return;
```

```
1183                 backing = src->reg->contains;
1184                 if (backing) {
1185                         /* Is it still valid? */
1186                         if (backing->reg != src->reg)
1187                                 backing = NULL;
1188                         else
1189                                 backing->reg = dest->reg;
1190                 }
1191                 dest->reg->contains = backing;
1192                 insn("mov", src, dest, NULL);
1193                 return;
1194         }

1196         /*
1197          * Are we moving to a register from a non-reg?
1198          *
1199          * See if we have the non-reg source already cached
1200          * in a register..
1201          */
1202         if (dest->type == STOR_REG) {
1203                 if (src->reg) {
1204                         struct reg_info *info = src->reg;
1205                         if (info->contains == src) {
1206                                 src = reginfo_reg(info);
1207                                 goto reg_reg_move;
1208                         }
1209                 }
1210                 dest->reg->contains = src;
1211                 src->reg = dest->reg;
1212         }

1214         if (src->type == STOR_REG) {
1215                 /* We could just mark the register dirty here and do lazy store.
1216                 src->reg->contains = dest;
1217                 dest->reg = src->reg;
1218         }

1220         if ((bits == 8) || (bits == 16)) {
1221                 if (is_dest)
1222                         opname = "mov";
1223                 else
1224                         opname = is_signed ? "movsx" : "movzx";
1225         } else
1226                 opname = "mov";

1228         insn(opbits(opname, bits), src, dest, comment);
1229 }

1231 static struct storage *emit_compare(struct expression *expr)
1232 {
1233         struct storage *left = x86_expression(expr->left);
1234         struct storage *right = x86_expression(expr->right);
1235         struct storage *reg1, *reg2;
1236         struct storage *new, *val;
1237         const char *opname = NULL;
1238         unsigned int right_bits = expr->right->ctype->bit_size;

1240         switch(expr->op) {
1241         case '<':               opname = "setl";        break;
1242         case '>':               opname = "setg";        break;
1243         case SPECIAL_LTE:
1244                                 opname = "setle";       break;
1245         case SPECIAL_GTE:
1246                                 opname = "setge";       break;
1247         case SPECIAL_EQUAL:     opname = "sete";        break;
1248         case SPECIAL_NOTEQUAL:  opname = "setne";       break;
```



```
1249         case SPECIAL_UNSIGNED_LT:
1250                                 opname = "setb";        break;
1251         case SPECIAL_UNSIGNED_GT:
1252                                 opname = "seta";        break;
1253         case SPECIAL_UNSIGNED_LTE:
1254                                 opname = "setb";        break;
1255         case SPECIAL_UNSIGNED_GTE:
1256                                 opname = "setae";       break;
1257         default:
1258                 assert(0);
1259                 break;
1260         }

1262         /* init EDX to 0 */
1263         val = new_storage(STOR_VALUE);
1264         val->flags = STOR_WANTS_FREE;

1266         reg1 = get_reg(&regclass_32_8);
1267         emit_move(val, reg1, NULL, NULL);

1269         /* move op1 into EAX */
1270         reg2 = get_reg_value(left, get_regclass(expr->left));

1272         /* perform comparison, RHS (op1, right) and LHS (op2, EAX) */
1273         insn(opbits("cmp", right_bits), right, reg2, NULL);
1274         put_reg(reg2);

1276         /* store result of operation, 0 or 1, in DL using SETcc */
1277         insn(opname, byte_reg(reg1), NULL, NULL);

1279         /* finally, store the result (DL) in a new pseudo / stack slot */
1280         new = stack_alloc(4);
1281         emit_move(reg1, new, NULL, "end EXPR_COMPARE");
1282         put_reg(reg1);

1284         return new;
1285 }

1287 static struct storage *emit_value(struct expression *expr)
1288 {
1289 #if 0 /* old and slow way */
1290         struct storage *new = stack_alloc(4);
1291         struct storage *val;

1293         val = new_storage(STOR_VALUE);
1294         val->value = (long long) expr->value;
1295         val->flags = STOR_WANTS_FREE;
1296         insn("movl", val, new, NULL);

1298         return new;
1299 #else
1300         struct storage *val;

1302         val = new_storage(STOR_VALUE);
1303         val->value = (long long) expr->value;

1305         return val;     /* FIXME: memory leak */
1306 #endif
1307 }

1309 static struct storage *emit_divide(struct expression *expr, struct storage *left
1310 {
1311         struct storage *eax_edx;
1312         struct storage *reg, *new;
1313         struct storage *val = new_storage(STOR_VALUE);
```

```
1315         emit_comment("begin DIVIDE");
1316         eax_edx = get_hardreg(hardreg_storage_table + EAX_EDX, 1);

1318         /* init EDX to 0 */
1319         val->flags = STOR_WANTS_FREE;
1320         emit_move(val, REG_EDX, NULL, NULL);

1322         new = stack_alloc(expr->ctype->bit_size / 8);

1324         /* EAX is dividend */
1325         emit_move(left, REG_EAX, NULL, NULL);

1327         reg = get_reg_value(right, &regclass_32);

1329         /* perform binop */
1330         insn("div", reg, REG_EAX, NULL);
1331         put_reg(reg);

1333         reg = REG_EAX;
1334         if (expr->op == '%')
1335                 reg = REG_EDX;
1336         emit_move(reg, new, NULL, NULL);

1338         put_reg(eax_edx);
1339         emit_comment("end DIVIDE");
1340         return new;
1341 }

1343 static struct storage *emit_binop(struct expression *expr)
1344 {
1345         struct storage *left = x86_expression(expr->left);
1346         struct storage *right = x86_expression(expr->right);
1347         struct storage *new;
1348         struct storage *dest, *src;
1349         const char *opname = NULL;
1350         const char *suffix = NULL;
1351         char opstr[16];
1352         int is_signed;

1354         /* Divides have special register constraints */
1355         if ((expr->op == '/') || (expr->op == '%'))
1356                 return emit_divide(expr, left, right);

1358         is_signed = type_is_signed(expr->ctype);

1360         switch (expr->op) {
1361         case '+':
1362                 opname = "add";
1363                 break;
1364         case '-':
1365                 opname = "sub";
1366                 break;
1367         case '&':
1368                 opname = "and";
1369                 break;
1370         case '|':
1371                 opname = "or";
1372                 break;
1373         case '^':
1374                 opname = "xor";
1375                 break;
1376         case SPECIAL_LEFTSHIFT:
1377                 opname = "shl";
1378                 break;
1379         case SPECIAL_RIGHTSHIFT:
1380                 if (is_signed)
```



```
1381                         opname = "sar";
1382                 else
1383                         opname = "shr";
1384                 break;
1385         case '*':
1386                 if (is_signed)
1387                         opname = "imul";
1388                 else
1389                         opname = "mul";
1390                 break;
1391         case SPECIAL_LOGICAL_AND:
1392                 warning(expr->pos, "bogus bitwise and for logical op (should use
1393                 opname = "and";
1394                 break;
1395         case SPECIAL_LOGICAL_OR:
1396                 warning(expr->pos, "bogus bitwise or for logical op (should use
1397                 opname = "or";
1398                 break;
1399         default:
1400                 error_die(expr->pos, "unhandled binop '%s'\n", show_special(expr
1401                 break;
1402         }

1404         dest = get_reg_value(right, &regclass_32);
1405         src = get_reg_value(left, &regclass_32);
1406         switch (expr->ctype->bit_size) {
1407         case 8:
1408                 suffix = "b";
1409                 break;
1410         case 16:
1411                 suffix = "w";
1412                 break;
1413         case 32:
1414                 suffix = "l";
1415                 break;
1416         case 64:
1417                 suffix = "q";           /* FIXME */
1418                 break;
1419         default:
1420                 assert(0);
1421                 break;
1422         }

1424         snprintf(opstr, sizeof(opstr), "%s%s", opname, suffix);

1426         /* perform binop */
1427         insn(opstr, src, dest, NULL);
1428         put_reg(src);

1430         /* store result in new pseudo / stack slot */
1431         new = stack_alloc(expr->ctype->bit_size / 8);
1432         emit_move(dest, new, NULL, "end EXPR_BINOP");

1434         put_reg(dest);

1436         return new;
1437 }

1439 static int emit_conditional_test(struct storage *val)
1440 {
1441         struct storage *reg;
1442         struct storage *target_val;
1443         int target_false;

1445         /* load result into EAX */
1446         emit_comment("begin if/conditional");
```

```
1447         reg = get_reg_value(val, &regclass_32);

1449         /* compare result with zero */
1450         insn("test", reg, reg, NULL);
1451         put_reg(reg);

1453         /* create conditional-failed label to jump to */
1454         target_false = new_label();
1455         target_val = new_storage(STOR_LABEL);
1456         target_val->label = target_false;
1457         target_val->flags = STOR_WANTS_FREE;
1458         insn("jz", target_val, NULL, NULL);

1460         return target_false;
1461 }

1463 static int emit_conditional_end(int target_false)
1464 {
1465         struct storage *cond_end_st;
1466         int cond_end;

1468         /* finished generating code for if-true statement.
1469          * add a jump-to-end jump to avoid falling through
1470          * to the if-false statement code.
1471          */
1472         cond_end = new_label();
1473         cond_end_st = new_storage(STOR_LABEL);
1474         cond_end_st->label = cond_end;
1475         cond_end_st->flags = STOR_WANTS_FREE;
1476         insn("jmp", cond_end_st, NULL, NULL);

1478         /* if we have both if-true and if-false statements,
1479          * the failed-conditional case will fall through to here
1480          */
1481         emit_label(target_false, NULL);

1483         return cond_end;
1484 }

1486 static void emit_if_conditional(struct statement *stmt)
1487 {
1488         struct storage *val;
1489         int cond_end;

1491         /* emit test portion of conditional */
1492         val = x86_expression(stmt->if_conditional);
1493         cond_end = emit_conditional_test(val);

1495         /* emit if-true statement */
1496         x86_statement(stmt->if_true);

1498         /* emit if-false statement, if present */
1499         if (stmt->if_false) {
1500                 cond_end = emit_conditional_end(cond_end);
1501                 x86_statement(stmt->if_false);
1502         }

1504         /* end of conditional; jump target for if-true branch */
1505         emit_label(cond_end, "end if");
1506 }

1508 static struct storage *emit_inc_dec(struct expression *expr, int postop)
1509 {
1510         struct storage *addr = x86_address_gen(expr->unop);
1511         struct storage *retval;
1512         char opname[16];
```



```
1514         strcpy(opname, opbits(expr->op == SPECIAL_INCREMENT ? "inc" : "dec",
1515                               expr->ctype->bit_size));

1517         if (postop) {
1518                 struct storage *new = stack_alloc(4);

1520                 emit_copy(new, addr, expr->unop->ctype);

1522                 retval = new;
1523         } else
1524                 retval = addr;

1526         insn(opname, addr, NULL, NULL);

1528         return retval;
1529 }

1531 static struct storage *emit_postop(struct expression *expr)
1532 {
1533         return emit_inc_dec(expr, 1);
1534 }

1536 static struct storage *emit_return_stmt(struct statement *stmt)
1537 {
1538         struct function *f = current_func;
1539         struct expression *expr = stmt->ret_value;
1540         struct storage *val = NULL, *jmplbl;

1542         if (expr && expr->ctype) {
1543                 val = x86_expression(expr);
1544                 assert(val != NULL);
1545                 emit_move(val, REG_EAX, expr->ctype, "return");
1546         }

1548         jmplbl = new_storage(STOR_LABEL);
1549         jmplbl->flags |= STOR_WANTS_FREE;
1550         jmplbl->label = f->ret_target;
1551         insn("jmp", jmplbl, NULL, NULL);

1553         return val;
1554 }

1556 static struct storage *emit_conditional_expr(struct expression *expr)
1557 {
1558         struct storage *cond, *true = NULL, *false = NULL;
1559         struct storage *new = stack_alloc(expr->ctype->bit_size / 8);
1560         int target_false, cond_end;

1562         /* evaluate conditional */
1563         cond = x86_expression(expr->conditional);
1564         target_false = emit_conditional_test(cond);

1566         /* handle if-true part of the expression */
1567         true = x86_expression(expr->cond_true);

1569         emit_copy(new, true, expr->ctype);

1571         cond_end = emit_conditional_end(target_false);

1573         /* handle if-false part of the expression */
1574         false = x86_expression(expr->cond_false);

1576         emit_copy(new, false, expr->ctype);

1578         /* end of conditional; jump target for if-true branch */
```

```
1579         emit_label(cond_end, "end conditional");

1581         return new;
1582 }

1584 static struct storage *emit_select_expr(struct expression *expr)
1585 {
1586         struct storage *cond = x86_expression(expr->conditional);
1587         struct storage *true = x86_expression(expr->cond_true);
1588         struct storage *false = x86_expression(expr->cond_false);
1589         struct storage *reg_cond, *reg_true, *reg_false;
1590         struct storage *new = stack_alloc(4);

1592         emit_comment("begin SELECT");
1593         reg_cond = get_reg_value(cond, get_regclass(expr->conditional));
1594         reg_true = get_reg_value(true, get_regclass(expr));
1595         reg_false = get_reg_value(false, get_regclass(expr));

1597         /*
1598          * Do the actual select: check the conditional for zero,
1599          * move false over true if zero
1600          */
1601         insn("test", reg_cond, reg_cond, NULL);
1602         insn("cmovz", reg_false, reg_true, NULL);

1604         /* Store it back */
1605         emit_move(reg_true, new, expr->ctype, NULL);
1606         put_reg(reg_cond);
1607         put_reg(reg_true);
1608         put_reg(reg_false);
1609         emit_comment("end SELECT");
1610         return new;
1611 }

1613 static struct storage *emit_symbol_expr_init(struct symbol *sym)
1614 {
1615         struct expression *expr = sym->initializer;
1616         struct symbol_private *priv = sym->aux;

1618         if (priv == NULL) {
1619                 priv = calloc(1, sizeof(*priv));
1620                 sym->aux = priv;

1622                 if (expr == NULL) {
1623                         struct storage *new = stack_alloc(4);
1624                         fprintf(stderr, "FIXME! no value for symbol %s.  creatin
1625                                 show_ident(sym->ident),
1626                                 new->pseudo, new->pseudo * 4);
1627                         priv->addr = new;
1628                 } else {
1629                         priv->addr = x86_expression(expr);
1630                 }
1631         }

1633         return priv->addr;
1634 }

1636 static struct storage *emit_string_expr(struct expression *expr)
1637 {
1638         struct function *f = current_func;
1639         int label = new_label();
1640         struct storage *new;

1642         push_cstring(f, expr->string, label);

1644         new = new_storage(STOR_LABEL);
```



```
1645         new->label = label;
1646         new->flags = STOR_LABEL_VAL | STOR_WANTS_FREE;
1647         return new;
1648 }

1650 static struct storage *emit_cast_expr(struct expression *expr)
1651 {
1652         struct symbol *old_type, *new_type;
1653         struct storage *op = x86_expression(expr->cast_expression);
1654         int oldbits, newbits;
1655         struct storage *new;

1657         old_type = expr->cast_expression->ctype;
1658         new_type = expr->cast_type;

1660         oldbits = old_type->bit_size;
1661         newbits = new_type->bit_size;
1662         if (oldbits >= newbits)
1663                 return op;

1665         emit_move(op, REG_EAX, old_type, "begin cast ..");

1667         new = stack_alloc(newbits / 8);
1668         emit_move(REG_EAX, new, new_type, ".... end cast");

1670         return new;
1671 }

1673 static struct storage *emit_regular_preop(struct expression *expr)
1674 {
1675         struct storage *target = x86_expression(expr->unop);
1676         struct storage *val, *new = stack_alloc(4);
1677         const char *opname = NULL;

1679         switch (expr->op) {
1680         case '!':
1681                 val = new_storage(STOR_VALUE);
1682                 val->flags = STOR_WANTS_FREE;
1683                 emit_move(val, REG_EDX, NULL, NULL);
1684                 emit_move(target, REG_EAX, expr->unop->ctype, NULL);
1685                 insn("test", REG_EAX, REG_EAX, NULL);
1686                 insn("setz", REG_DL, NULL, NULL);
1687                 emit_move(REG_EDX, new, expr->unop->ctype, NULL);

1689                 break;
1690         case '~':
1691                 opname = "not";
1692         case '-':
1693                 if (!opname)
1694                         opname = "neg";
1695                 emit_move(target, REG_EAX, expr->unop->ctype, NULL);
1696                 insn(opname, REG_EAX, NULL, NULL);
1697                 emit_move(REG_EAX, new, expr->unop->ctype, NULL);
1698                 break;
1699         default:
1700                 assert(0);
1701                 break;
1702         }

1704         return new;
1705 }

1707 static void emit_case_statement(struct statement *stmt)
1708 {
1709         emit_labelsym(stmt->case_label, NULL);
1710         x86_statement(stmt->case_statement);
```

```
}

static void emit_switch_statement(struct statement *stmt)
{
        struct storage *val = x86_expression(stmt->switch_expression);
        struct symbol *sym, *default_sym = NULL;
        struct storage *labelsym, *label;
        int switch_end = 0;

        emit_move(val, REG_EAX, stmt->switch_expression->ctype, "begin case");

        /*
         * This is where a _real_ back-end would go through the
         * cases to decide whether to use a lookup table or a
         * series of comparisons etc
         */
        FOR_EACH_PTR(stmt->switch_case->symbol_list, sym) {
                struct statement *case_stmt = sym->stmt;
                struct expression *expr = case_stmt->case_expression;
                struct expression *to = case_stmt->case_to;

                /* default: */
                if (!expr)
                        default_sym = sym;

                /* case NNN: */
                else {
                        struct storage *case_val = new_val(expr->value);

                        assert (expr->type == EXPR_VALUE);

                        insn("cmpl", case_val, REG_EAX, NULL);

                        if (!to) {
                                labelsym = new_labelsym(sym);
                                insn("je", labelsym, NULL, NULL);
                        } else {
                                int next_test;

                                label = new_storage(STOR_LABEL);
                                label->flags |= STOR_WANTS_FREE;
                                label->label = next_test = new_label();

                                /* FIXME: signed/unsigned */
                                insn("jl", label, NULL, NULL);

                                case_val = new_val(to->value);
                                insn("cmpl", case_val, REG_EAX, NULL);

                                /* TODO: implement and use refcounting... */
                                label = new_storage(STOR_LABEL);
                                label->flags |= STOR_WANTS_FREE;
                                label->label = next_test;

                                /* FIXME: signed/unsigned */
                                insn("jg", label, NULL, NULL);

                                labelsym = new_labelsym(sym);
                                insn("jmp", labelsym, NULL, NULL);

                                emit_label(next_test, NULL);
                        }
                }
        } END_FOR_EACH_PTR(sym);

        if (default_sym) {
```



```
                labelsym = new_labelsym(default_sym);
                insn("jmp", labelsym, NULL, "default");
        } else {
                label = new_storage(STOR_LABEL);
                label->flags |= STOR_WANTS_FREE;
                label->label = switch_end = new_label();
                insn("jmp", label, NULL, "goto end of switch");
        }

        x86_statement(stmt->switch_statement);

        if (stmt->switch_break->used)
                emit_labelsym(stmt->switch_break, NULL);

        if (switch_end)
                emit_label(switch_end, NULL);
}

static void x86_struct_member(struct symbol *sym)
{
        printf("\t%s:%d:%ld at offset %ld.%d", show_ident(sym->ident), sym->bit_
        printf("\n");
}

static void x86_symbol(struct symbol *sym)
{
        struct symbol *type;

        if (!sym)
                return;

        type = sym->ctype.base_type;
        if (!type)
                return;

        /*
         * Show actual implementation information
         */
        switch (type->type) {

        case SYM_ARRAY:
                if (sym->initializer)
                        emit_array(sym);
                else
                        emit_array_noinit(sym);
                break;

        case SYM_BASETYPE:
                if (sym->initializer) {
                        emit_object_pre(show_ident(sym->ident),
                                        sym->ctype.modifiers,
                                        sym->ctype.alignment,
                                        sym->bit_size / 8);
                        emit_scalar(sym->initializer, sym->bit_size);
                        stor_sym_init(sym);
                } else
                        emit_scalar_noinit(sym);
                break;

        case SYM_STRUCT:
        case SYM_UNION: {
                struct symbol *member;

                printf(" {\n");
                FOR_EACH_PTR(type->symbol_list, member) {
                        x86_struct_member(member);
```

```
1843                 } END_FOR_EACH_PTR(member);
1844                 printf("}\n");
1845                 break;
1846         }

1848         case SYM_FN: {
1849                 struct statement *stmt = type->stmt;
1850                 if (stmt) {
1851                         emit_func_pre(sym);
1852                         x86_statement(stmt);
1853                         emit_func_post(sym);
1854                 }
1855                 break;
1856         }

1858         default:
1859                 break;
1860         }

1862         if (sym->initializer && (type->type != SYM_BASETYPE) &&
1863             (type->type != SYM_ARRAY)) {
1864                 printf(" = \n");
1865                 x86_expression(sym->initializer);
1866         }
1867 }

1869 static void x86_symbol_init(struct symbol *sym);

1871 static void x86_symbol_decl(struct symbol_list *syms)
1872 {
1873         struct symbol *sym;
1874         FOR_EACH_PTR(syms, sym) {
1875                 x86_symbol_init(sym);
1876         } END_FOR_EACH_PTR(sym);
1877 }

1879 static void loopstk_push(int cont_lbl, int loop_bottom_lbl)
1880 {
1881         struct function *f = current_func;
1882         struct loop_stack *ls;

1884         ls = malloc(sizeof(*ls));
1885         ls->continue_lbl = cont_lbl;
1886         ls->loop_bottom_lbl = loop_bottom_lbl;
1887         ls->next = f->loop_stack;
1888         f->loop_stack = ls;
1889 }

1891 static void loopstk_pop(void)
1892 {
1893         struct function *f = current_func;
1894         struct loop_stack *ls;

1896         assert(f->loop_stack != NULL);
1897         ls = f->loop_stack;
1898         f->loop_stack = f->loop_stack->next;
1899         free(ls);
1900 }

1902 static int loopstk_break(void)
1903 {
1904         return current_func->loop_stack->loop_bottom_lbl;
1905 }

1907 static int loopstk_continue(void)
1908 {
```



```
1909         return current_func->loop_stack->continue_lbl;
1910 }

1912 static void emit_loop(struct statement *stmt)
1913 {
1914         struct statement  *pre_statement = stmt->iterator_pre_statement;
1915         struct expression *pre_condition = stmt->iterator_pre_condition;
1916         struct statement  *statement = stmt->iterator_statement;
1917         struct statement  *post_statement = stmt->iterator_post_statement;
1918         struct expression *post_condition = stmt->iterator_post_condition;
1919         int loop_top = 0, loop_bottom, loop_continue;
1920         int have_bottom = 0;
1921         struct storage *val;

1923         loop_bottom = new_label();
1924         loop_continue = new_label();
1925         loopstk_push(loop_continue, loop_bottom);

1927         x86_symbol_decl(stmt->iterator_syms);
1928         x86_statement(pre_statement);
1929         if (!post_condition || post_condition->type != EXPR_VALUE || post_condit
1930                 loop_top = new_label();
1931                 emit_label(loop_top, "loop top");
1932         }
1933         if (pre_condition) {
1934                 if (pre_condition->type == EXPR_VALUE) {
1935                         if (!pre_condition->value) {
1936                                 struct storage *lbv;
1937                                 lbv = new_storage(STOR_LABEL);
1938                                 lbv->label = loop_bottom;
1939                                 lbv->flags = STOR_WANTS_FREE;
1940                                 insn("jmp", lbv, NULL, "go to loop bottom");
1941                                 have_bottom = 1;
1942                         }
1943                 } else {
1944                         struct storage *lbv = new_storage(STOR_LABEL);
1945                         lbv->label = loop_bottom;
1946                         lbv->flags = STOR_WANTS_FREE;
1947                         have_bottom = 1;

1949                         val = x86_expression(pre_condition);

1951                         emit_move(val, REG_EAX, NULL, "loop pre condition");
1952                         insn("test", REG_EAX, REG_EAX, NULL);
1953                         insn("jz", lbv, NULL, NULL);
1954                 }
1955         }
1956         x86_statement(statement);
1957         if (stmt->iterator_continue->used)
1958                 emit_label(loop_continue, "'continue' iterator");
1959         x86_statement(post_statement);
1960         if (!post_condition) {
1961                 struct storage *lbv = new_storage(STOR_LABEL);
1962                 lbv->label = loop_top;
1963                 lbv->flags = STOR_WANTS_FREE;
1964                 insn("jmp", lbv, NULL, "go to loop top");
1965         } else if (post_condition->type == EXPR_VALUE) {
1966                 if (post_condition->value) {
1967                         struct storage *lbv = new_storage(STOR_LABEL);
1968                         lbv->label = loop_top;
1969                         lbv->flags = STOR_WANTS_FREE;
1970                         insn("jmp", lbv, NULL, "go to loop top");
1971                 }
1972         } else {
1973                 struct storage *lbv = new_storage(STOR_LABEL);
1974                 lbv->label = loop_top;
```

```
1975                lbv->flags = STOR_WANTS_FREE;

1977                val = x86_expression(post_condition);

1979                emit_move(val, REG_EAX, NULL, "loop post condition");
1980                insn("test", REG_EAX, REG_EAX, NULL);
1981                insn("jnz", lbv, NULL, NULL);
1982        }
1983        if (have_bottom || stmt->iterator_break->used)
1984                emit_label(loop_bottom, "loop bottom");

1986        loopstk_pop();
1987 }

1989 /*
1990  * Print out a statement
1991  */
1992 static struct storage *x86_statement(struct statement *stmt)
1993 {
1994        if (!stmt)
1995                return NULL;
1996        switch (stmt->type) {
1997        default:
1998                return NULL;
1999        case STMT_RETURN:
2000                return emit_return_stmt(stmt);
2001        case STMT_DECLARATION:
2002                x86_symbol_decl(stmt->declaration);
2003                break;
2004        case STMT_COMPOUND: {
2005                struct statement *s;
2006                struct storage *last = NULL;

2008                FOR_EACH_PTR(stmt->stmts, s) {
2009                        last = x86_statement(s);
2010                } END_FOR_EACH_PTR(s);

2012                return last;
2013        }

2015        case STMT_EXPRESSION:
2016                return x86_expression(stmt->expression);
2017        case STMT_IF:
2018                emit_if_conditional(stmt);
2019                return NULL;

2021        case STMT_CASE:
2022                emit_case_statement(stmt);
2023                break;
2024        case STMT_SWITCH:
2025                emit_switch_statement(stmt);
2026                break;

2028        case STMT_ITERATOR:
2029                emit_loop(stmt);
2030                break;

2032        case STMT_NONE:
2033                break;

2035        case STMT_LABEL:
2036                printf(".L%p:\n", stmt->label_identifier);
2037                x86_statement(stmt->label_statement);
2038                break;

2040        case STMT_GOTO:
```



```
2041                if (stmt->goto_expression) {
2042                        struct storage *val = x86_expression(stmt->goto_expressi
2043                        printf("\tgoto *v%d\n", val->pseudo);
2044                } else if (!strcmp("break", show_ident(stmt->goto_label->ident))
2045                        struct storage *lbv = new_storage(STOR_LABEL);
2046                        lbv->label = loopstk_break();
2047                        lbv->flags = STOR_WANTS_FREE;
2048                        insn("jmp", lbv, NULL, "'break'; go to loop bottom");
2049                } else if (!strcmp("continue", show_ident(stmt->goto_label->iden
2050                        struct storage *lbv = new_storage(STOR_LABEL);
2051                        lbv->label = loopstk_continue();
2052                        lbv->flags = STOR_WANTS_FREE;
2053                        insn("jmp", lbv, NULL, "'continue'; go to loop top");
2054                } else {
2055                        struct storage *labelsym = new_labelsym(stmt->goto_label
2056                        insn("jmp", labelsym, NULL, NULL);
2057                }
2058                break;
2059        case STMT_ASM:
2060                printf("\tasm( .... )\n");
2061                break;
2062        }
2063        return NULL;
2064 }

2066 static struct storage *x86_call_expression(struct expression *expr)
2067 {
2068        struct function *f = current_func;
2069        struct symbol *direct;
2070        struct expression *arg, *fn;
2071        struct storage *retval, *fncall;
2072        int framesize;
2073        char s[64];

2075        if (!expr->ctype) {
2076                warning(expr->pos, "\tcall with no type!");
2077                return NULL;
2078        }

2080        framesize = 0;
2081        FOR_EACH_PTR_REVERSE(expr->args, arg) {
2082                struct storage *new = x86_expression(arg);
2083                int size = arg->ctype->bit_size;

2085                /*
2086                 * FIXME: i386 SysV ABI dictates that values
2087                 * smaller than 32 bits should be placed onto
2088                 * the stack as 32-bit objects.  We should not
2089                 * blindly do a 32-bit push on objects smaller
2090                 * than 32 bits.
2091                 */
2092                if (size < 32)
2093                        size = 32;
2094                insn("pushl", new, NULL,
2095                     !framesize ? "begin function call" : NULL);

2097                framesize += bits_to_bytes(size);
2098        } END_FOR_EACH_PTR_REVERSE(arg);

2100        fn = expr->fn;

2102        /* Remove dereference, if any */
2103        direct = NULL;
2104        if (fn->type == EXPR_PREOP) {
2105                if (fn->unop->type == EXPR_SYMBOL) {
2106                        struct symbol *sym = fn->unop->symbol;
```

```
2107                         if (sym->ctype.base_type->type == SYM_FN)
2108                                 direct = sym;
2109                 }
2110         }
2111         if (direct) {
2112                 struct storage *direct_stor = new_storage(STOR_SYM);
2113                 direct_stor->flags |= STOR_WANTS_FREE;
2114                 direct_stor->sym = direct;
2115                 insn("call", direct_stor, NULL, NULL);
2116         } else {
2117                 fncall = x86_expression(fn);
2118                 emit_move(fncall, REG_EAX, fn->ctype, NULL);

2120                 strcpy(s, "\tcall\t*%eax\n");
2121                 push_text_atom(f, s);
2122         }

2124         /* FIXME: pay attention to BITS_IN_POINTER */
2125         if (framesize) {
2126                 struct storage *val = new_storage(STOR_VALUE);
2127                 val->value = (long long) framesize;
2128                 val->flags = STOR_WANTS_FREE;
2129                 insn("addl", val, REG_ESP, NULL);
2130         }

2132         retval = stack_alloc(4);
2133         emit_move(REG_EAX, retval, NULL, "end function call");

2135         return retval;
2136 }

2138 static struct storage *x86_address_gen(struct expression *expr)
2139 {
2140         struct function *f = current_func;
2141         struct storage *addr;
2142         struct storage *new;
2143         char s[32];

2145         addr = x86_expression(expr->unop);
2146         if (expr->unop->type == EXPR_SYMBOL)
2147                 return addr;

2149         emit_move(addr, REG_EAX, NULL, "begin deref ..");

2151         /* FIXME: operand size */
2152         strcpy(s, "\tmovl\t(%eax), %ecx\n");
2153         push_text_atom(f, s);

2155         new = stack_alloc(4);
2156         emit_move(REG_ECX, new, NULL, ".... end deref");

2158         return new;
2159 }

2161 static struct storage *x86_assignment(struct expression *expr)
2162 {
2163         struct expression *target = expr->left;
2164         struct storage *val, *addr;

2166         if (!expr->ctype)
2167                 return NULL;

2169         val = x86_expression(expr->right);
2170         addr = x86_address_gen(target);

2172         switch (val->type) {
```



```
2173         /* copy, where both operands are memory */
2174         case STOR_PSEUDO:
2175         case STOR_ARG:
2176                 emit_copy(addr, val, expr->ctype);
2177                 break;

2179         /* copy, one or zero operands are memory */
2180         case STOR_REG:
2181         case STOR_SYM:
2182         case STOR_VALUE:
2183         case STOR_LABEL:
2184                 emit_move(val, addr, expr->left->ctype, NULL);
2185                 break;

2187         case STOR_LABELSYM:
2188                 assert(0);
2189                 break;
2190         }
2191         return val;
2192 }

2194 static int x86_initialization(struct symbol *sym, struct expression *expr)
2195 {
2196         struct storage *val, *addr;
2197         int bits;

2199         if (!expr->ctype)
2200                 return 0;

2202         bits = expr->ctype->bit_size;
2203         val = x86_expression(expr);
2204         addr = x86_symbol_expr(sym);
2205         // FIXME! The "target" expression is for bitfield store information.
2206         // Leave it NULL, which works fine.
2207         emit_store(NULL, addr, val, bits);
2208         return 0;
2209 }

2211 static struct storage *x86_access(struct expression *expr)
2212 {
2213         return x86_address_gen(expr);
2214 }

2216 static struct storage *x86_preop(struct expression *expr)
2217 {
2218         /*
2219          * '*' is an lvalue access, and is fundamentally different
2220          * from an arithmetic operation. Maybe it should have an
2221          * expression type of its own..
2222          */
2223         if (expr->op == '*')
2224                 return x86_access(expr);
2225         if (expr->op == SPECIAL_INCREMENT || expr->op == SPECIAL_DECREMENT)
2226                 return emit_inc_dec(expr, 0);
2227         return emit_regular_preop(expr);
2228 }

2230 static struct storage *x86_symbol_expr(struct symbol *sym)
2231 {
2232         struct storage *new = stack_alloc(4);

2234         if (sym->ctype.modifiers & (MOD_TOPLEVEL | MOD_EXTERN | MOD_STATIC)) {
2235                 printf("\tmovi.%d\t\tv%d,$%s\n", bits_in_pointer, new->pseudo, s
2236                 return new;
2237         }
2238         if (sym->ctype.modifiers & MOD_ADDRESSABLE) {
```

```
2239                 printf("\taddi.%d\t\tv%d,vFP,$%lld\n", bits_in_pointer, new->pse
2240                 return new;
2241         }
2242         printf("\taddi.%d\t\tv%d,vFP,$offsetof(%s:%p)\n", bits_in_pointer, new->
2243         return new;
2244 }

2246 static void x86_symbol_init(struct symbol *sym)
2247 {
2248         struct symbol_private *priv = sym->aux;
2249         struct expression *expr = sym->initializer;
2250         struct storage *new;

2252         if (expr)
2253                 new = x86_expression(expr);
2254         else
2255                 new = stack_alloc(sym->bit_size / 8);

2257         if (!priv) {
2258                 priv = calloc(1, sizeof(*priv));
2259                 sym->aux = priv;
2260                 /* FIXME: leak! we don't free... */
2261                 /* (well, we don't free symbols either) */
2262         }

2264         priv->addr = new;
2265 }

2267 static int type_is_signed(struct symbol *sym)
2268 {
2269         if (sym->type == SYM_NODE)
2270                 sym = sym->ctype.base_type;
2271         if (sym->type == SYM_PTR)
2272                 return 0;
2273         return !(sym->ctype.modifiers & MOD_UNSIGNED);
2274 }

2276 static struct storage *x86_label_expr(struct expression *expr)
2277 {
2278         struct storage *new = stack_alloc(4);
2279         printf("\tmovi.%d\t\tv%d,.L%p\n", bits_in_pointer, new->pseudo, expr->la
2280         return new;
2281 }

2283 static struct storage *x86_statement_expr(struct expression *expr)
2284 {
2285         return x86_statement(expr->statement);
2286 }

2288 static int x86_position_expr(struct expression *expr, struct symbol *base)
2289 {
2290         struct storage *new = x86_expression(expr->init_expr);
2291         struct symbol *ctype = expr->init_expr->ctype;

2293         printf("\tinsert v%d at [%d:%d] of %s\n", new->pseudo,
2294                 expr->init_offset, ctype->bit_offset,
2295                 show_ident(base->ident));
2296         return 0;
2297 }

2299 static void x86_initializer_expr(struct expression *expr, struct symbol *ctype)
2300 {
2301         struct expression *entry;

2303         FOR_EACH_PTR(expr->expr_list, entry) {
2304                 // Nested initializers have their positions already
```



```
2305                 // recursively calculated - just output them too
2306                 if (entry->type == EXPR_INITIALIZER) {
2307                         x86_initializer_expr(entry, ctype);
2308                         continue;
2309                 }

2311                 // Ignore initializer indexes and identifiers - the
2312                 // evaluator has taken them into account
2313                 if (entry->type == EXPR_IDENTIFIER || entry->type == EXPR_INDEX)
2314                         continue;
2315                 if (entry->type == EXPR_POS) {
2316                         x86_position_expr(entry, ctype);
2317                         continue;
2318                 }
2319                 x86_initialization(ctype, entry);
2320         } END_FOR_EACH_PTR(entry);
2321 }

2323 /*
2324  * Print out an expression. Return the pseudo that contains the
2325  * variable.
2326  */
2327 static struct storage *x86_expression(struct expression *expr)
2328 {
2329         if (!expr)
2330                 return NULL;

2332         if (!expr->ctype) {
2333                 struct position *pos = &expr->pos;
2334                 printf("\tno type at %s:%d:%d\n",
2335                         stream_name(pos->stream),
2336                         pos->line, pos->pos);
2337                 return NULL;
2338         }

2340         switch (expr->type) {
2341         default:
2342                 return NULL;
2343         case EXPR_CALL:
2344                 return x86_call_expression(expr);

2346         case EXPR_ASSIGNMENT:
2347                 return x86_assignment(expr);

2349         case EXPR_COMPARE:
2350                 return emit_compare(expr);
2351         case EXPR_BINOP:
2352         case EXPR_COMMA:
2353         case EXPR_LOGICAL:
2354                 return emit_binop(expr);
2355         case EXPR_PREOP:
2356                 return x86_preop(expr);
2357         case EXPR_POSTOP:
2358                 return emit_postop(expr);
2359         case EXPR_SYMBOL:
2360                 return emit_symbol_expr_init(expr->symbol);
2361         case EXPR_DEREF:
2362         case EXPR_SIZEOF:
2363         case EXPR_ALIGNOF:
2364                 warning(expr->pos, "invalid expression after evaluation");
2365                 return NULL;
2366         case EXPR_CAST:
2367         case EXPR_FORCE_CAST:
2368         case EXPR_IMPLIED_CAST:
2369                 return emit_cast_expr(expr);
2370         case EXPR_VALUE:
```

```
2371                return emit_value(expr);
2372        case EXPR_STRING:
2373                return emit_string_expr(expr);
2374        case EXPR_INITIALIZER:
2375                x86_initializer_expr(expr, expr->ctype);
2376                return NULL;
2377        case EXPR_SELECT:
2378                return emit_select_expr(expr);
2379        case EXPR_CONDITIONAL:
2380                return emit_conditional_expr(expr);
2381        case EXPR_STATEMENT:
2382                return x86_statement_expr(expr);
2383        case EXPR_LABEL:
2384                return x86_label_expr(expr);

2386        // None of these should exist as direct expressions: they are only
2387        // valid as sub-expressions of initializers.
2388        case EXPR_POS:
2389                warning(expr->pos, "unable to show plain initializer position ex
2390                return NULL;
2391        case EXPR_IDENTIFIER:
2392                warning(expr->pos, "unable to show identifier expression");
2393                return NULL;
2394        case EXPR_INDEX:
2395                warning(expr->pos, "unable to show index expression");
2396                return NULL;
2397        case EXPR_TYPE:
2398                warning(expr->pos, "unable to show type expression");
2399                return NULL;
2400        case EXPR_FVALUE:
2401                warning(expr->pos, "floating point support is not implemented");
2402                return NULL;
2403        }
2404        return NULL;
2405 }
```

```
**********************************************************
    2459 Fri Dec 21 15:00:10 2018
new/usr/src/tools/smatch/src/compile.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Example trivial client program that uses the sparse library
 * and x86 backend.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 * Copyright 2003 Jeff Garzik
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"
#include "compile.h"

static void clean_up_symbols(struct symbol_list *list)
{
        struct symbol *sym;

        FOR_EACH_PTR(list, sym) {
                expand_symbol(sym);
                emit_one_symbol(sym);
        } END_FOR_EACH_PTR(sym);
}

int main(int argc, char **argv)
{
        char *file;
        struct string_list *filelist = NULL;

        bits_in_bool = 8;
```



```
        clean_up_symbols(sparse_initialize(argc, argv, &filelist));
        FOR_EACH_PTR_NOTAG(filelist, file) {
                struct symbol_list *list;
                const char *basename = strrchr(file, '/');
                basename = basename ?  basename+1 : file;

                list = sparse(file);

                // Do type evaluation and simplification
                emit_unit_begin(basename);
                clean_up_symbols(list);
                emit_unit_end();
        } END_FOR_EACH_PTR_NOTAG(file);

#if 0
        // And show the allocation statistics
        show_ident_alloc();
        show_token_alloc();
        show_symbol_alloc();
        show_expression_alloc();
        show_statement_alloc();
        show_string_alloc();
        show_bytes_alloc();
#endif
        return 0;
}
```

```
**********************************************************
     199 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/compile.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef COMPILE_H
#define COMPILE_H

struct symbol;

extern void emit_one_symbol(struct symbol *);
extern void emit_unit_begin(const char *);
extern void emit_unit_end(void);

#endif /* COMPILE_H */
```

```
**********************************************************
    9069 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cse.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * CSE - walk the linearized instruction flow, and
 * see if we can simplify it and apply CSE on it.
 *
 * Copyright (C) 2004 Linus Torvalds
 */

#include <string.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <assert.h>

#include "parse.h"
#include "expression.h"
#include "linearize.h"
#include "flow.h"

#define INSN_HASH_SIZE 256
static struct instruction_list *insn_hash_table[INSN_HASH_SIZE];

int repeat_phase;

static int phi_compare(pseudo_t phi1, pseudo_t phi2)
{
        const struct instruction *def1 = phi1->def;
        const struct instruction *def2 = phi2->def;

        if (def1->src1 != def2->src1)
                return def1->src1 < def2->src1 ? -1 : 1;
        if (def1->bb != def2->bb)
                return def1->bb < def2->bb ? -1 : 1;
        return 0;
}


static void clean_up_one_instruction(struct basic_block *bb, struct instruction
{
        unsigned long hash;

        if (!insn->bb)
                return;
        assert(insn->bb == bb);
        repeat_phase |= simplify_instruction(insn);
        if (!insn->bb)
                return;
        hash = (insn->opcode << 3) + (insn->size >> 3);
        switch (insn->opcode) {
        case OP_SEL:
                hash += hashval(insn->src3);
                /* Fall through */

        /* Binary arithmetic */
        case OP_ADD: case OP_SUB:
        case OP_MULU: case OP_MULS:
        case OP_DIVU: case OP_DIVS:
        case OP_MODU: case OP_MODS:
        case OP_SHL:
        case OP_LSR: case OP_ASR:
```



```
        case OP_AND: case OP_OR:

        /* Binary logical */
        case OP_XOR: case OP_AND_BOOL:
        case OP_OR_BOOL:

        /* Binary comparison */
        case OP_SET_EQ: case OP_SET_NE:
        case OP_SET_LE: case OP_SET_GE:
        case OP_SET_LT: case OP_SET_GT:
        case OP_SET_B:  case OP_SET_A:
        case OP_SET_BE: case OP_SET_AE:
                hash += hashval(insn->src2);
                /* Fall through */

        /* Unary */
        case OP_NOT: case OP_NEG:
                hash += hashval(insn->src1);
                break;

        case OP_SETVAL:
                hash += hashval(insn->val);
                break;

        case OP_SYMADDR:
                hash += hashval(insn->symbol);
                break;

        case OP_CAST:
        case OP_SCAST:
        case OP_PTRCAST:
                /*
                 * This is crap! Many "orig_types" are the
                 * same as far as casts go, we should generate
                 * some kind of "type hash" that is identical
                 * for identical casts
                 */
                hash += hashval(insn->orig_type);
                hash += hashval(insn->src);
                break;

        /* Other */
        case OP_PHI: {
                pseudo_t phi;
                FOR_EACH_PTR(insn->phi_list, phi) {
                        struct instruction *def;
                        if (phi == VOID || !phi->def)
                                continue;
                        def = phi->def;
                        hash += hashval(def->src1);
                        hash += hashval(def->bb);
                } END_FOR_EACH_PTR(phi);
                break;
        }

        default:
                /*
                 * Nothing to do, don't even bother hashing them,
                 * we're not going to try to CSE them
                 */
                return;
        }
        hash += hash >> 16;
        hash &= INSN_HASH_SIZE-1;
        add_instruction(insn_hash_table + hash, insn);
}
```

```
128 static void clean_up_insns(struct entrypoint *ep)
129 {
130         struct basic_block *bb;
131
132         FOR_EACH_PTR(ep->bbs, bb) {
133                 struct instruction *insn;
134                 FOR_EACH_PTR(bb->insns, insn) {
135                         clean_up_one_instruction(bb, insn);
136                 } END_FOR_EACH_PTR(insn);
137         } END_FOR_EACH_PTR(bb);
138 }
139
140 /* Compare two (sorted) phi-lists */
141 static int phi_list_compare(struct pseudo_list *l1, struct pseudo_list *l2)
142 {
143         pseudo_t phi1, phi2;
144
145         PREPARE_PTR_LIST(l1, phi1);
146         PREPARE_PTR_LIST(l2, phi2);
147         for (;;) {
148                 int cmp;
149
150                 while (phi1 && (phi1 == VOID || !phi1->def))
151                         NEXT_PTR_LIST(phi1);
152                 while (phi2 && (phi2 == VOID || !phi2->def))
153                         NEXT_PTR_LIST(phi2);
154
155                 if (!phi1)
156                         return phi2 ? -1 : 0;
157                 if (!phi2)
158                         return phi1 ? 1 : 0;
159                 cmp = phi_compare(phi1, phi2);
160                 if (cmp)
161                         return cmp;
162                 NEXT_PTR_LIST(phi1);
163                 NEXT_PTR_LIST(phi2);
164         }
165         /* Not reached, but we need to make the nesting come out right */
166         FINISH_PTR_LIST(phi2);
167         FINISH_PTR_LIST(phi1);
168 }
169
170 static int insn_compare(const void *_i1, const void *_i2)
171 {
172         const struct instruction *i1 = _i1;
173         const struct instruction *i2 = _i2;
174
175         if (i1->opcode != i2->opcode)
176                 return i1->opcode < i2->opcode ? -1 : 1;
177
178         switch (i1->opcode) {
179
180         /* commutative binop */
181         case OP_ADD:
182         case OP_MULU: case OP_MULS:
183         case OP_AND_BOOL: case OP_OR_BOOL:
184         case OP_AND: case OP_OR:
185         case OP_XOR:
186         case OP_SET_EQ: case OP_SET_NE:
187                 if (i1->src1 == i2->src2 && i1->src2 == i2->src1)
188                         return 0;
189                 goto case_binops;
190
191         case OP_SEL:
192                 if (i1->src3 != i2->src3)
```



```
193                         return i1->src3 < i2->src3 ? -1 : 1;
194                 /* Fall-through to binops */
195
196         /* Binary arithmetic */
197         case OP_SUB:
198         case OP_DIVU: case OP_DIVS:
199         case OP_MODU: case OP_MODS:
200         case OP_SHL:
201         case OP_LSR: case OP_ASR:
202
203         /* Binary comparison */
204         case OP_SET_LE: case OP_SET_GE:
205         case OP_SET_LT: case OP_SET_GT:
206         case OP_SET_B:  case OP_SET_A:
207         case OP_SET_BE: case OP_SET_AE:
208         case_binops:
209                 if (i1->src2 != i2->src2)
210                         return i1->src2 < i2->src2 ? -1 : 1;
211                 /* Fall through to unops */
212
213         /* Unary */
214         case OP_NOT: case OP_NEG:
215                 if (i1->src1 != i2->src1)
216                         return i1->src1 < i2->src1 ? -1 : 1;
217                 break;
218
219         case OP_SYMADDR:
220                 if (i1->symbol != i2->symbol)
221                         return i1->symbol < i2->symbol ? -1 : 1;
222                 break;
223
224         case OP_SETVAL:
225                 if (i1->val != i2->val)
226                         return i1->val < i2->val ? -1 : 1;
227                 break;
228
229         /* Other */
230         case OP_PHI:
231                 return phi_list_compare(i1->phi_list, i2->phi_list);
232
233         case OP_CAST:
234         case OP_SCAST:
235         case OP_PTRCAST:
236                 /*
237                  * This is crap! See the comments on hashing.
238                  */
239                 if (i1->orig_type != i2->orig_type)
240                         return i1->orig_type < i2->orig_type ? -1 : 1;
241                 if (i1->src != i2->src)
242                         return i1->src < i2->src ? -1 : 1;
243                 break;
244
245         default:
246                 warning(i1->pos, "bad instruction on hash chain");
247         }
248         if (i1->size != i2->size)
249                 return i1->size < i2->size ? -1 : 1;
250         return 0;
251 }
252
253 static void sort_instruction_list(struct instruction_list **list)
254 {
255         sort_list((struct ptr_list **)list , insn_compare);
256 }
257
258 static struct instruction * cse_one_instruction(struct instruction *insn, struct
```

```
259 {
260         convert_instruction_target(insn, def->target);
261
262         kill_instruction(insn);
263         repeat_phase |= REPEAT_CSE;
264         return def;
265 }
266
267 /*
268  * Does "bb1" dominate "bb2"?
269  */
270 static int bb_dominates(struct entrypoint *ep, struct basic_block *bb1, struct b
271 {
272         struct basic_block *parent;
273
274         /* Nothing dominates the entrypoint.. */
275         if (bb2 == ep->entry->bb)
276                 return 0;
277         FOR_EACH_PTR(bb2->parents, parent) {
278                 if (parent == bb1)
279                         continue;
280                 if (parent->generation == generation)
281                         continue;
282                 parent->generation = generation;
283                 if (!bb_dominates(ep, bb1, parent, generation))
284                         return 0;
285         } END_FOR_EACH_PTR(parent);
286         return 1;
287 }
288
289 static struct basic_block *trivial_common_parent(struct basic_block *bb1, struct
290 {
291         struct basic_block *parent;
292
293         if (bb_list_size(bb1->parents) != 1)
294                 return NULL;
295         parent = first_basic_block(bb1->parents);
296         if (bb_list_size(bb2->parents) != 1)
297                 return NULL;
298         if (first_basic_block(bb2->parents) != parent)
299                 return NULL;
300         return parent;
301 }
302
303 static inline void remove_instruction(struct instruction_list **list, struct ins
304 {
305         delete_ptr_list_entry((struct ptr_list **)list, insn, count);
306 }
307
308 static void add_instruction_to_end(struct instruction *insn, struct basic_block
309 {
310         struct instruction *br = delete_last_instruction(&bb->insns);
311         insn->bb = bb;
312         add_instruction(&bb->insns, insn);
313         add_instruction(&bb->insns, br);
314 }
315
316 static struct instruction * try_to_cse(struct entrypoint *ep, struct instruction
317 {
318         struct basic_block *b1, *b2, *common;
319
320         /*
321          * OK, i1 and i2 are the same instruction, modulo "target".
322          * We should now see if we can combine them.
323          */
324         b1 = i1->bb;
```



```
325         b2 = i2->bb;
326
327         /*
328          * Currently we only handle the uninteresting degenerate case where
329          * the CSE is inside one basic-block.
330          */
331         if (b1 == b2) {
332                 struct instruction *insn;
333                 FOR_EACH_PTR(b1->insns, insn) {
334                         if (insn == i1)
335                                 return cse_one_instruction(i2, i1);
336                         if (insn == i2)
337                                 return cse_one_instruction(i1, i2);
338                 } END_FOR_EACH_PTR(insn);
339                 warning(b1->pos, "Whaa? unable to find CSE instructions");
340                 return i1;
341         }
342         if (bb_dominates(ep, b1, b2, ++bb_generation))
343                 return cse_one_instruction(i2, i1);
344
345         if (bb_dominates(ep, b2, b1, ++bb_generation))
346                 return cse_one_instruction(i1, i2);
347
348         /* No direct dominance - but we could try to find a common ancestor.. */
349         common = trivial_common_parent(b1, b2);
350         if (common) {
351                 i1 = cse_one_instruction(i2, i1);
352                 remove_instruction(&b1->insns, i1, 1);
353                 add_instruction_to_end(i1, common);
354         }
355
356         return i1;
357 }
358
359 void cleanup_and_cse(struct entrypoint *ep)
360 {
361         int i;
362
363         simplify_memops(ep);
364 repeat:
365         repeat_phase = 0;
366         clean_up_insns(ep);
367         if (repeat_phase & REPEAT_CFG_CLEANUP)
368                 kill_unreachable_bbs(ep);
369         for (i = 0; i < INSN_HASH_SIZE; i++) {
370                 struct instruction_list **list = insn_hash_table + i;
371                 if (*list) {
372                         if (instruction_list_size(*list) > 1) {
373                                 struct instruction *insn, *last;
374
375                                 sort_instruction_list(list);
376
377                                 last = NULL;
378                                 FOR_EACH_PTR(*list, insn) {
379                                         if (!insn->bb)
380                                                 continue;
381                                         if (last) {
382                                                 if (!insn_compare(last, insn))
383                                                         insn = try_to_cse(ep, la
384                                         }
385                                         last = insn;
386                                 } END_FOR_EACH_PTR(insn);
387                         }
388                         free_ptr_list((struct ptr_list **)list);
389                 }
390         }
```

```
392         if (repeat_phase & REPEAT_SYMBOL_CLEANUP)
393                 simplify_memops(ep);

395         if (repeat_phase & REPEAT_CSE)
396                 goto repeat;
397 }
```

```
**********************************************************
    5733 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/ctags.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Sparse Ctags
 *
 * Ctags generates tags from preprocessing results.
 *
 * Copyright (C) 2006 Christopher Li
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>

#include "parse.h"
#include "scope.h"

static struct symbol_list *taglist = NULL;

static void examine_symbol(struct symbol *sym);

#define MAX(_x,_y) ((_x) > (_y) ? (_x) : (_y))

static int cmp_sym(const void *m, const void *n)
{
        const struct ident *a = ((const struct symbol *)m)->ident;
        const struct ident *b = ((const struct symbol *)n)->ident;
        int ret = strncmp(a->name, b->name, MAX(a->len, b->len));
        if (!ret) {
                const struct position a_pos = ((const struct symbol *)m)->pos;
                const struct position b_pos = ((const struct symbol *)n)->pos;

                ret = strcmp(stream_name(a_pos.stream),
                             stream_name(b_pos.stream));
                if (!ret)
                        return a_pos.line < b_pos.line;
        }
        return ret;
}

static void show_tag_header(FILE *fp)
{
        fprintf(fp, "!_TAG_FILE_FORMAT\t2\t/extended format; --format=1 will not
```



```
        fprintf(fp, "!_TAG_FILE_SORTED\t0\t/0=unsorted, 1=sorted, 2=foldcase/\n"
        fprintf(fp, "!_TAG_PROGRAM_AUTHOR\tChristopher Li\t/sparse@chrisli.org/\
        fprintf(fp, "!_TAG_PROGRAM_NAME\tSparse Ctags\t//\n");
        fprintf(fp, "!_TAG_PROGRAM_URL\thttp://www.kernel.org/pub/software/devel
        fprintf(fp, "!_TAG_PROGRAM_VERSION\t0.01\t//\n");
}

static inline void show_symbol_tag(FILE *fp, struct symbol *sym)
{
        fprintf(fp, "%s\t%s\t%d;\"\t%c\tfile:\n", show_ident(sym->ident),
               stream_name(sym->pos.stream), sym->pos.line, (int)sym->kind);
}

static void show_tags(struct symbol_list *list)
{
        struct symbol *sym;
        struct ident *ident = NULL;
        struct position pos = {};
        static const char *filename;
        FILE *fp;

        if (!list)
                return;

        fp = fopen("tags", "w");
        if (!fp) {
                perror("open tags file");
                return;
        }
        show_tag_header(fp);
        FOR_EACH_PTR(list, sym) {
                if (ident == sym->ident && pos.line == sym->pos.line &&
                    !strcmp(filename, stream_name(sym->pos.stream)))
                        continue;

                show_symbol_tag(fp, sym);
                ident = sym->ident;
                pos = sym->pos;
                filename = stream_name(sym->pos.stream);
        } END_FOR_EACH_PTR(sym);
        fclose(fp);
}

static inline void add_tag(struct symbol *sym)
{
        if (sym->ident && !sym->visited) {
                sym->visited = 1;
                add_symbol(&taglist, sym);
        }
}

static inline void examine_members(struct symbol_list *list)
{
        struct symbol *sym;

        FOR_EACH_PTR(list, sym) {
                sym->kind = 'm';
                examine_symbol(sym);
        } END_FOR_EACH_PTR(sym);
}

static void examine_symbol(struct symbol *sym)
{
        struct symbol *base = sym;

        if (!sym || sym->visited)
```

```
127                 return;
128         if (sym->ident && sym->ident->reserved)
129                 return;
130         if (sym->type == SYM_KEYWORD || sym->type == SYM_PREPROCESSOR)
131                 return;

133         add_tag(sym);
134         base = sym->ctype.base_type;

136         switch (sym->type) {
137         case SYM_NODE:
138                 if (base->type == SYM_FN)
139                         sym->kind = 'f';
140                 examine_symbol(base);
141                 break;
142         case SYM_STRUCT:
143                 sym->kind = 's';
144                 examine_members(sym->symbol_list);
145                 break;
146         case SYM_UNION:
147                 sym->kind = 'u';
148                 examine_members(sym->symbol_list);
149                 break;
150         case SYM_ENUM:
151                 sym->kind = 'e';
152         case SYM_PTR:
153         case SYM_TYPEOF:
154         case SYM_BITFIELD:
155         case SYM_FN:
156         case SYM_ARRAY:
157                 examine_symbol(sym->ctype.base_type);
158                 break;
159         case SYM_BASETYPE:
160                 break;

162         default:
163                 die("unknown symbol %s namespace:%d type:%d\n", show_ident(sym->
164                     sym->namespace, sym->type);
165         }
166         if (!sym->kind)
167                 sym->kind = 'v';
168         return;
169 }

171 static void examine_namespace(struct symbol *sym)
172 {
173         if (sym->visited)
174                 return;
175         if (sym->ident && sym->ident->reserved)
176                 return;

178         switch(sym->namespace) {
179         case NS_KEYWORD:
180         case NS_PREPROCESSOR:
181                 return;
182         case NS_LABEL:
183                 sym->kind = 'l';
184                 break;
185         case NS_MACRO:
186         case NS_UNDEF:
187                 sym->kind = 'd';
188                 break;
189         case NS_TYPEDEF:
190                 sym->kind = 't';
191         case NS_SYMBOL:
192         case NS_STRUCT:
```



```
193                 examine_symbol(sym);
194                 break;
195         default:
196                 die("unknown namespace %d symbol:%s type:%d\n", sym->namespace,
197                     show_ident(sym->ident), sym->type);
198         }
199         add_tag(sym);
200 }

202 static inline void examine_symbol_list(struct symbol_list *list)
203 {
204         struct symbol *sym;

206         if (!list)
207                 return;
208         FOR_EACH_PTR(list, sym) {
209                 examine_namespace(sym);
210         } END_FOR_EACH_PTR(sym);
211 }

213 int main(int argc, char **argv)
214 {
215         struct string_list *filelist = NULL;
216         char *file;

218         examine_symbol_list(sparse_initialize(argc, argv, &filelist));
219         FOR_EACH_PTR_NOTAG(filelist, file) {
220                 sparse(file);
221                 examine_symbol_list(file_scope->symbols);
222         } END_FOR_EACH_PTR_NOTAG(file);
223         examine_symbol_list(global_scope->symbols);
224         sort_list((struct ptr_list **)&taglist, cmp_sym);
225         show_tags(taglist);
226         return 0;
227 }
```

**********
647 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
tester: hashtable.o tester.o hashtable_itr.o
        gcc -g -Wall -O -o tester hashtable.o hashtable_itr.o tester.o -lm

all: tester old_tester

tester.o:       tester.c
        gcc -g -Wall -O -c tester.c -o tester.o

old_tester: hashtable_powers.o tester.o hashtable_itr.o
        gcc -g -Wall -O -o old_tester hashtable_powers.o hashtable_itr.o tester.

hashtable_powers.o:     hashtable_powers.c
        gcc -g -Wall -O -c hashtable_powers.c -o hashtable_powers.o

hashtable.o:    hashtable.c
        gcc -g -Wall -O -c hashtable.c -o hashtable.o

hashtable_itr.o: hashtable_itr.c
        gcc -g -Wall -O -c hashtable_itr.c -o hashtable_itr.o

tidy:
        rm *.o

clean: tidy
        rm -f tester old_tester
```

```
**********************************************************
    9031 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/hashtable.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Copyright (C) 2004 Christopher Clark <firstname.lastname@cl.cam.ac.uk> */

#include "hashtable.h"
#include "hashtable_private.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

/*
Credit for primes table: Aaron Krowne
 http://br.endernet.org/~akrowne/
 http://planetmath.org/encyclopedia/GoodHashTablePrimes.html
*/
static const unsigned int primes[] = {
53, 97, 193, 389,
769, 1543, 3079, 6151,
12289, 24593, 49157, 98317,
196613, 393241, 786433, 1572869,
3145739, 6291469, 12582917, 25165843,
50331653, 100663319, 201326611, 402653189,
805306457, 1610612741
};
const unsigned int prime_table_length = sizeof(primes)/sizeof(primes[0]);
const float max_load_factor = 0.65;

/*****************************************************************************/
struct hashtable *
create_hashtable(unsigned int minsize,
                 unsigned int (*hashf) (void*),
                 int (*eqf) (void*,void*))
{
    struct hashtable *h;
    unsigned int pindex, size = primes[0];
    /* Check requested hashtable isn't too large */
    if (minsize > (1u << 30)) return NULL;
    /* Enforce size as prime */
    for (pindex=0; pindex < prime_table_length; pindex++) {
        if (primes[pindex] > minsize) { size = primes[pindex]; break; }
    }
    h = (struct hashtable *)malloc(sizeof(struct hashtable));
    if (NULL == h) return NULL; /*oom*/
    h->table = (struct entry **)malloc(sizeof(struct entry*) * size);
    if (NULL == h->table) { free(h); return NULL; } /*oom*/
    memset(h->table, 0, size * sizeof(struct entry *));
    h->tablelength  = size;
    h->primeindex   = pindex;
    h->entrycount   = 0;
    h->hashfn       = hashf;
    h->eqfn         = eqf;
    h->loadlimit    = (unsigned int) ceil(size * max_load_factor);
    return h;
}

/*****************************************************************************/
unsigned int
hash(struct hashtable *h, void *k)
{
    /* Aim to protect against poor hash functions by adding logic here
     * - logic taken from java 1.4 hashtable source */
```



```
    unsigned int i = h->hashfn(k);
    i += ~(i << 9);
    i ^=  ((i >> 14) | (i << 18)); /* >>> */
    i +=  (i << 4);
    i ^=  ((i >> 10) | (i << 22)); /* >>> */
    return i;
}

/*****************************************************************************/
static int
hashtable_expand(struct hashtable *h)
{
    /* Double the size of the table to accomodate more entries */
    struct entry **newtable;
    struct entry *e;
    struct entry **pE;
    unsigned int newsize, i, index;
    /* Check we're not hitting max capacity */
    if (h->primeindex == (prime_table_length - 1)) return 0;
    newsize = primes[++(h->primeindex)];

    newtable = (struct entry **)malloc(sizeof(struct entry*) * newsize);
    if (NULL != newtable)
    {
        memset(newtable, 0, newsize * sizeof(struct entry *));
        /* This algorithm is not 'stable'. ie. it reverses the list
         * when it transfers entries between the tables */
        for (i = 0; i < h->tablelength; i++) {
            while (NULL != (e = h->table[i])) {
                h->table[i] = e->next;
                index = indexFor(newsize,e->h);
                e->next = newtable[index];
                newtable[index] = e;
            }
        }
        free(h->table);
        h->table = newtable;
    }
    /* Plan B: realloc instead */
    else
    {
        newtable = (struct entry **)
                   realloc(h->table, newsize * sizeof(struct entry *));
        if (NULL == newtable) { (h->primeindex)--; return 0; }
        h->table = newtable;
        memset(newtable[h->tablelength], 0, newsize - h->tablelength);
        for (i = 0; i < h->tablelength; i++) {
            for (pE = &(newtable[i]), e = *pE; e != NULL; e = *pE) {
                index = indexFor(newsize,e->h);
                if (index == i)
                {
                    pE = &(e->next);
                }
                else
                {
                    *pE = e->next;
                    e->next = newtable[index];
                    newtable[index] = e;
                }
            }
        }
    }
    h->tablelength = newsize;
    h->loadlimit   = (unsigned int) ceil(newsize * max_load_factor);
    return -1;
}
```

```
/*****************************************************************************/
unsigned int
hashtable_count(struct hashtable *h)
{
    return h->entrycount;
}

/*****************************************************************************/
int
hashtable_insert(struct hashtable *h, void *k, void *v)
{
    /* This method allows duplicate keys - but they shouldn't be used */
    unsigned int index;
    struct entry *e;
    if (++(h->entrycount) > h->loadlimit)
    {
        /* Ignore the return value. If expand fails, we should
         * still try cramming just this value into the existing table
         * -- we may not have memory for a larger table, but one more
         * element may be ok. Next time we insert, we'll try expanding again.*/
        hashtable_expand(h);
    }
    e = (struct entry *)malloc(sizeof(struct entry));
    if (NULL == e) { --(h->entrycount); return 0; } /*oom*/
    e->h = hash(h,k);
    index = indexFor(h->tablelength,e->h);
    e->k = k;
    e->v = v;
    e->next = h->table[index];
    h->table[index] = e;
    return -1;
}

/*****************************************************************************/
void * /* returns value associated with key */
hashtable_search(struct hashtable *h, void *k)
{
    struct entry *e;
    unsigned int hashvalue, index;
    hashvalue = hash(h,k);
    index = indexFor(h->tablelength,hashvalue);
    e = h->table[index];
    while (NULL != e)
    {
        /* Check hash value to short circuit heavier comparison */
        if ((hashvalue == e->h) && (h->eqfn(k, e->k))) return e->v;
        e = e->next;
    }
    return NULL;
}

/*****************************************************************************/
void * /* returns value associated with key */
hashtable_remove(struct hashtable *h, void *k)
{
    /* TODO: consider compacting the table when the load factor drops enough,
     *       or provide a 'compact' method. */

    struct entry *e;
    struct entry **pE;
    void *v;
    unsigned int hashvalue, index;

    hashvalue = hash(h,k);
    index = indexFor(h->tablelength,hash(h,k));
```



```
    pE = &(h->table[index]);
    e = *pE;
    while (NULL != e)
    {
        /* Check hash value to short circuit heavier comparison */
        if ((hashvalue == e->h) && (h->eqfn(k, e->k)))
        {
            *pE = e->next;
            h->entrycount--;
            v = e->v;
            freekey(e->k);
            free(e);
            return v;
        }
        pE = &(e->next);
        e = e->next;
    }
    return NULL;
}

/*****************************************************************************/
/* destroy */
void
hashtable_destroy(struct hashtable *h, int free_values)
{
    unsigned int i;
    struct entry *e, *f;
    struct entry **table = h->table;
    if (free_values)
    {
        for (i = 0; i < h->tablelength; i++)
        {
            e = table[i];
            while (NULL != e)
            { f = e; e = e->next; freekey(f->k); free(f->v); free(f); }
        }
    }
    else
    {
        for (i = 0; i < h->tablelength; i++)
        {
            e = table[i];
            while (NULL != e)
            { f = e; e = e->next; freekey(f->k); free(f); }
        }
    }
    free(h->table);
    free(h);
}

/*
 * Copyright (c) 2002, Christopher Clark
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * * Neither the name of the original author; nor the names of any contributors
```

```
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

```
**********************************************************
    7401 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/hashtable.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Copyright (C) 2002 Christopher Clark <firstname.lastname@cl.cam.ac.uk> */

#ifndef __HASHTABLE_CWC22_H__
#define __HASHTABLE_CWC22_H__

struct hashtable;

/* Example of use:
 *
 *      struct hashtable  *h;
 *      struct some_key   *k;
 *      struct some_value *v;
 *
 *      static unsigned int         hash_from_key_fn( void *k );
 *      static int                  keys_equal_fn ( void *key1, void *key2 );
 *
 *      h = create_hashtable(16, hash_from_key_fn, keys_equal_fn);
 *      k = (struct some_key *)     malloc(sizeof(struct some_key));
 *      v = (struct some_value *)   malloc(sizeof(struct some_value));
 *
 *      (initialise k and v to suitable values)
 *
 *      if (! hashtable_insert(h,k,v) )
 *      {     exit(-1);               }
 *
 *      if (NULL == (found = hashtable_search(h,k) ))
 *      {    printf("not found!");                  }
 *
 *      if (NULL == (found = hashtable_remove(h,k) ))
 *      {    printf("Not found\n");                 }
 *
 */

/* Macros may be used to define type-safe(r) hashtable access functions, with
 * methods specialized to take known key and value types as parameters.
 *
 * Example:
 *
 * Insert this at the start of your file:
 *
 * DEFINE_HASHTABLE_INSERT(insert_some, struct some_key, struct some_value);
 * DEFINE_HASHTABLE_SEARCH(search_some, struct some_key, struct some_value);
 * DEFINE_HASHTABLE_REMOVE(remove_some, struct some_key, struct some_value);
 *
 * This defines the functions 'insert_some', 'search_some' and 'remove_some'.
 * These operate just like hashtable_insert etc., with the same parameters,
 * but their function signatures have 'struct some_key *' rather than
 * 'void *', and hence can generate compile time errors if your program is
 * supplying incorrect data as a key (and similarly for value).
 *
 * Note that the hash and key equality functions passed to create_hashtable
 * still take 'void *' parameters instead of 'some key *'. This shouldn't be
 * a difficult issue as they're only defined and passed once, and the other
 * functions will ensure that only valid keys are supplied to them.
 *
 * The cost for this checking is increased code size and runtime overhead
 * - if performance is important, it may be worth switching back to the
 * unsafe methods once your program has been debugged with the safe methods.
 * This just requires switching to some simple alternative defines - eg:
 * #define insert_some hashtable_insert
```



```
 *
 */

/*****************************************************************************
 * create_hashtable

 * @name                    create_hashtable
 * @param   minsize         minimum initial size of hashtable
 * @param   hashfunction    function for hashing keys
 * @param   key_eq_fn       function for determining key equality
 * @return                  newly created hashtable or NULL on failure
 */

struct hashtable *
create_hashtable(unsigned int minsize,
                 unsigned int (*hashfunction) (void*),
                 int (*key_eq_fn) (void*,void*));

/*****************************************************************************
 * hashtable_insert

 * @name        hashtable_insert
 * @param   h   the hashtable to insert into
 * @param   k   the key - hashtable claims ownership and will free on removal
 * @param   v   the value - does not claim ownership
 * @return      non-zero for successful insertion
 *
 * This function will cause the table to expand if the insertion would take
 * the ratio of entries to table size over the maximum load factor.
 *
 * This function does not check for repeated insertions with a duplicate key.
 * The value returned when using a duplicate key is undefined -- when
 * the hashtable changes size, the order of retrieval of duplicate key
 * entries is reversed.
 * If in doubt, remove before insert.
 */

int
hashtable_insert(struct hashtable *h, void *k, void *v);

#define DEFINE_HASHTABLE_INSERT(fnname, keytype, valuetype) \
int fnname (struct hashtable *h, keytype *k, valuetype *v) \
{ \
    return hashtable_insert(h,k,v); \
}

/*****************************************************************************
 * hashtable_search

 * @name        hashtable_search
 * @param   h   the hashtable to search
 * @param   k   the key to search for  - does not claim ownership
 * @return      the value associated with the key, or NULL if none found
 */

void *
hashtable_search(struct hashtable *h, void *k);

#define DEFINE_HASHTABLE_SEARCH(fnname, keytype, valuetype) \
valuetype * fnname (struct hashtable *h, keytype *k) \
{ \
    return (valuetype *) (hashtable_search(h,k)); \
}

/*****************************************************************************
 * hashtable_remove
```

```
127
128  * @name        hashtable_remove
129  * @param   h   the hashtable to remove the item from
130  * @param   k   the key to search for  - does not claim ownership
131  * @return      the value associated with the key, or NULL if none found
132  */

134 void * /* returns value */
135 hashtable_remove(struct hashtable *h, void *k);

137 #define DEFINE_HASHTABLE_REMOVE(fnname, keytype, valuetype) \
138 valuetype * fnname (struct hashtable *h, keytype *k) \
139 { \
140     return (valuetype *) (hashtable_remove(h,k)); \
141 }


144 /*****************************************************************************
145  * hashtable_count
146  *
147  * @name        hashtable_count
148  * @param   h   the hashtable
149  * @return      the number of items stored in the hashtable
150  */
151 unsigned int
152 hashtable_count(struct hashtable *h);


155 /*****************************************************************************
156  * hashtable_destroy
157  *
158  * @name        hashtable_destroy
159  * @param   h   the hashtable
160  * @param       free_values     whether to call 'free' on the remaining values
161  */

163 void
164 hashtable_destroy(struct hashtable *h, int free_values);

166 #endif /* __HASHTABLE_CWC22_H__ */

168 /*
169  * Copyright (c) 2002, Christopher Clark
170  * All rights reserved.
171  *
172  * Redistribution and use in source and binary forms, with or without
173  * modification, are permitted provided that the following conditions
174  * are met:
175  *
176  * * Redistributions of source code must retain the above copyright
177  * notice, this list of conditions and the following disclaimer.
178  *
179  * * Redistributions in binary form must reproduce the above copyright
180  * notice, this list of conditions and the following disclaimer in the
181  * documentation and/or other materials provided with the distribution.
182  *
183  * * Neither the name of the original author; nor the names of any contributors
184  * may be used to endorse or promote products derived from this software
185  * without specific prior written permission.
186  *
187  *
188  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
189  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
190  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
191  * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
192  * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
```



```
193  * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
194  * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
195  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
196  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
197  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
198  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
199 */
```

```
**********************************************************
    5789 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/hashtable_itr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 /* Copyright (C) 2002, 2004 Christopher Clark  <firstname.lastname@cl.cam.ac.uk>

  3 #include "hashtable.h"
  4 #include "hashtable_private.h"
  5 #include "hashtable_itr.h"
  6 #include <stdlib.h> /* defines NULL */

  8 /*****************************************************************************/
  9 /* hashtable_iterator    - iterator constructor */

 11 struct hashtable_itr *
 12 hashtable_iterator(struct hashtable *h)
 13 {
 14     unsigned int i, tablelength;
 15     struct hashtable_itr *itr = (struct hashtable_itr *)
 16         malloc(sizeof(struct hashtable_itr));
 17     if (NULL == itr) return NULL;
 18     itr->h = h;
 19     itr->e = NULL;
 20     itr->parent = NULL;
 21     tablelength = h->tablelength;
 22     itr->index = tablelength;
 23     if (0 == h->entrycount) return itr;

 25     for (i = 0; i < tablelength; i++)
 26     {
 27         if (NULL != h->table[i])
 28         {
 29             itr->e = h->table[i];
 30             itr->index = i;
 31             break;
 32         }
 33     }
 34     return itr;
 35 }

 37 /*****************************************************************************/
 38 /* key      - return the key of the (key,value) pair at the current position */
 39 /* value    - return the value of the (key,value) pair at the current position *

 41 void *
 42 hashtable_iterator_key(struct hashtable_itr *i)
 43 { return i->e->k; }

 45 void *
 46 hashtable_iterator_value(struct hashtable_itr *i)
 47 { return i->e->v; }

 49 /*****************************************************************************/
 50 /* advance - advance the iterator to the next element
 51  *           returns zero if advanced to end of table */

 53 int
 54 hashtable_iterator_advance(struct hashtable_itr *itr)
 55 {
 56     unsigned int j,tablelength;
 57     struct entry **table;
 58     struct entry *next;
 59     if (NULL == itr->e) return 0; /* stupidity check */
```



```
 61     next = itr->e->next;
 62     if (NULL != next)
 63     {
 64         itr->parent = itr->e;
 65         itr->e = next;
 66         return -1;
 67     }
 68     tablelength = itr->h->tablelength;
 69     itr->parent = NULL;
 70     if (tablelength <= (j = ++(itr->index)))
 71     {
 72         itr->e = NULL;
 73         return 0;
 74     }
 75     table = itr->h->table;
 76     while (NULL == (next = table[j]))
 77     {
 78         if (++j >= tablelength)
 79         {
 80             itr->index = tablelength;
 81             itr->e = NULL;
 82             return 0;
 83         }
 84     }
 85     itr->index = j;
 86     itr->e = next;
 87     return -1;
 88 }

 90 /*****************************************************************************/
 91 /* remove - remove the entry at the current iterator position
 92  *          and advance the iterator, if there is a successive
 93  *          element.
 94  *          If you want the value, read it before you remove:
 95  *          beware memory leaks if you don't.
 96  *          Returns zero if end of iteration. */

 98 int
 99 hashtable_iterator_remove(struct hashtable_itr *itr)
100 {
101     struct entry *remember_e, *remember_parent;
102     int ret;

104     /* Do the removal */
105     if (NULL == (itr->parent))
106     {
107         /* element is head of a chain */
108         itr->h->table[itr->index] = itr->e->next;
109     } else {
110         /* element is mid-chain */
111         itr->parent->next = itr->e->next;
112     }
113     /* itr->e is now outside the hashtable */
114     remember_e = itr->e;
115     itr->h->entrycount--;
116     freekey(remember_e->k);

118     /* Advance the iterator, correcting the parent */
119     remember_parent = itr->parent;
120     ret = hashtable_iterator_advance(itr);
121     if (itr->parent == remember_e) { itr->parent = remember_parent; }
122     free(remember_e);
123     return ret;
124 }

126 /*****************************************************************************/
```

```
127 int /* returns zero if not found */
128 hashtable_iterator_search(struct hashtable_itr *itr,
129                           struct hashtable *h, void *k)
130 {
131     struct entry *e, *parent;
132     unsigned int hashvalue, index;
133
134     hashvalue = hash(h,k);
135     index = indexFor(h->tablelength,hashvalue);
136
137     e = h->table[index];
138     parent = NULL;
139     while (NULL != e)
140     {
141         /* Check hash value to short circuit heavier comparison */
142         if ((hashvalue == e->h) && (h->eqfn(k, e->k)))
143         {
144             itr->index = index;
145             itr->e = e;
146             itr->parent = parent;
147             itr->h = h;
148             return -1;
149         }
150         parent = e;
151         e = e->next;
152     }
153     return 0;
154 }
155
156
157 /*
158  * Copyright (c) 2002, 2004, Christopher Clark
159  * All rights reserved.
160  *
161  * Redistribution and use in source and binary forms, with or without
162  * modification, are permitted provided that the following conditions
163  * are met:
164  *
165  * * Redistributions of source code must retain the above copyright
166  * notice, this list of conditions and the following disclaimer.
167  *
168  * * Redistributions in binary form must reproduce the above copyright
169  * notice, this list of conditions and the following disclaimer in the
170  * documentation and/or other materials provided with the distribution.
171  *
172  * * Neither the name of the original author; nor the names of any contributors
173  * may be used to endorse or promote products derived from this software
174  * without specific prior written permission.
175  *
176  *
177  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
178  * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
179  * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
180  * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
181  * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
182  * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
183  * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
184  * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
185  * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
186  * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
187  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
188  */
```

```
**********************************************************
    4092 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/hashtable_itr.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Copyright (C) 2002, 2004 Christopher Clark <firstname.lastname@cl.cam.ac.uk>

#ifndef __HASHTABLE_ITR_CWC22__
#define __HASHTABLE_ITR_CWC22__
#include "hashtable.h"
#include "hashtable_private.h" /* needed to enable inlining */

/*****************************************************************************/
/* This struct is only concrete here to allow the inlining of two of the
 * accessor functions. */
struct hashtable_itr
{
    struct hashtable *h;
    struct entry *e;
    struct entry *parent;
    unsigned int index;
};


/*****************************************************************************/
/* hashtable_iterator
 */

struct hashtable_itr *
hashtable_iterator(struct hashtable *h);

/*****************************************************************************/
/* hashtable_iterator_key
 * - return the value of the (key,value) pair at the current position */

extern inline void *
hashtable_iterator_key(struct hashtable_itr *i)
{
    return i->e->k;
}

/*****************************************************************************/
/* value - return the value of the (key,value) pair at the current position */

extern inline void *
hashtable_iterator_value(struct hashtable_itr *i)
{
    return i->e->v;
}

/*****************************************************************************/
/* advance - advance the iterator to the next element
 *           returns zero if advanced to end of table */

int
hashtable_iterator_advance(struct hashtable_itr *itr);

/*****************************************************************************/
/* remove - remove current element and advance the iterator to the next element
 *          NB: if you need the value to free it, read it before
 *          removing. ie: beware memory leaks!
 *          returns zero if advanced to end of table */

int
hashtable_iterator_remove(struct hashtable_itr *itr);
```



```
/*****************************************************************************/
/* search - overwrite the supplied iterator, to point to the entry
 *          matching the supplied key.
 *          h points to the hashtable to be searched.
 *          returns zero if not found. */
int
hashtable_iterator_search(struct hashtable_itr *itr,
                          struct hashtable *h, void *k);

#define DEFINE_HASHTABLE_ITERATOR_SEARCH(fnname, keytype) \
int fnname (struct hashtable_itr *i, struct hashtable *h, keytype *k) \
{ \
    return (hashtable_iterator_search(i,h,k)); \
}



#endif /* __HASHTABLE_ITR_CWC22__*/

/*
 * Copyright (c) 2002, 2004, Christopher Clark
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * * Neither the name of the original author; nor the names of any contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```
**********************************************************
    2961 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/hashtable_private.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Copyright (C) 2002, 2004 Christopher Clark <firstname.lastname@cl.cam.ac.uk>

#ifndef __HASHTABLE_PRIVATE_CWC22_H__
#define __HASHTABLE_PRIVATE_CWC22_H__

#include "hashtable.h"

/*****************************************************************************/
struct entry
{
    void *k, *v;
    unsigned int h;
    struct entry *next;
};

struct hashtable {
    unsigned int tablelength;
    struct entry **table;
    unsigned int entrycount;
    unsigned int loadlimit;
    unsigned int primeindex;
    unsigned int (*hashfn) (void *k);
    int (*eqfn) (void *k1, void *k2);
};

/*****************************************************************************/
unsigned int
hash(struct hashtable *h, void *k);

/*****************************************************************************/
/* indexFor */
static inline unsigned int
indexFor(unsigned int tablelength, unsigned int hashvalue) {
    return (hashvalue % tablelength);
};

/* Only works if tablelength == 2^N */
/*static inline unsigned int
indexFor(unsigned int tablelength, unsigned int hashvalue)
{
    return (hashvalue & (tablelength - 1u));
}
*/

/*****************************************************************************/
#define freekey(X) free(X)
/*define freekey(X) ; */


/*****************************************************************************/

#endif /* __HASHTABLE_PRIVATE_CWC22_H__*/

/*
 * Copyright (c) 2002, Christopher Clark
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
```



```
 *
 * * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * * Neither the name of the original author; nor the names of any contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```
**********************************************************
    2581 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/hashtable_utility.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Copyright (C) 2002 Christopher Clark <firstname.lastname@cl.cam.ac.uk> */

#include "hashtable.h"
#include "hashtable_private.h"
#include "hashtable_utility.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

/*****************************************************************************/
/* hashtable_change
 *
 * function to change the value associated with a key, where there already
 * exists a value bound to the key in the hashtable.
 * Source due to Holger Schemel.
 *
 *  */
int
hashtable_change(struct hashtable *h, void *k, void *v)
{
    struct entry *e;
    unsigned int hashvalue, index;
    hashvalue = hash(h,k);
    index = indexFor(h->tablelength,hashvalue);
    e = h->table[index];
    while (NULL != e)
    {
        /* Check hash value to short circuit heavier comparison */
        if ((hashvalue == e->h) && (h->eqfn(k, e->k)))
        {
            free(e->v);
            e->v = v;
            return -1;
        }
        e = e->next;
    }
    return 0;
}

/*
 * Copyright (c) 2002, Christopher Clark
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * * Neither the name of the original author; nor the names of any contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
```



```
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```
**********************************************************
    2213 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/hashtable_utility.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Copyright (C) 2002 Christopher Clark <firstname.lastname@cl.cam.ac.uk> */

#ifndef __HASHTABLE_CWC22_UTILITY_H__
#define __HASHTABLE_CWC22_UTILITY_H__

/*****************************************************************************
 * hashtable_change
 *
 * function to change the value associated with a key, where there already
 * exists a value bound to the key in the hashtable.
 * Source due to Holger Schemel.
 *
 * @name        hashtable_change
 * @param   h   the hashtable
 * @param       key
 * @param       value
 *
 */
int
hashtable_change(struct hashtable *h, void *k, void *v);

#endif /* __HASHTABLE_CWC22_H__ */

/*
 * Copyright (c) 2002, Christopher Clark
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * * Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * * Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in the
 * documentation and/or other materials provided with the distribution.
 *
 * * Neither the name of the original author; nor the names of any contributors
 * may be used to endorse or promote products derived from this software
 * without specific prior written permission.
 *
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */
```

```
**********************************************************
    8577 Fri Dec 21 15:00:11 2018
new/usr/src/tools/smatch/src/cwchash/tester.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 /* Copyright (C) 2002, 2004 Christopher Clark <firstname.lastname@cl.cam.ac.uk>

  3 #include "hashtable.h"
  4 #include "hashtable_itr.h"
  5 #include <stdlib.h>
  6 #include <stdio.h>
  7 #include <string.h> /* for memcmp */

  9 static const int ITEM_COUNT = 4000;

 11 typedef unsigned int uint32_t;
 12 typedef unsigned short uint16_t;

 14 /*****************************************************************************/
 15 struct key
 16 {
 17     uint32_t one_ip; uint32_t two_ip; uint16_t one_port; uint16_t two_port;
 18 };

 20 struct value
 21 {
 22     char *id;
 23 };

 25 DEFINE_HASHTABLE_INSERT(insert_some, struct key, struct value);
 26 DEFINE_HASHTABLE_SEARCH(search_some, struct key, struct value);
 27 DEFINE_HASHTABLE_REMOVE(remove_some, struct key, struct value);
 28 DEFINE_HASHTABLE_ITERATOR_SEARCH(search_itr_some, struct key);


 31 /*****************************************************************************/
 32 static unsigned int
 33 hashfromkey(void *ky)
 34 {
 35     struct key *k = (struct key *)ky;
 36     return (((k->one_ip << 17) | (k->one_ip >> 15)) ^ k->two_ip) +
 37             (k->one_port * 17) + (k->two_port * 13 * 29);
 38 }

 40 static int
 41 equalkeys(void *k1, void *k2)
 42 {
 43     return (0 == memcmp(k1,k2,sizeof(struct key)));
 44 }

 46 /*****************************************************************************/
 47 int
 48 main(int argc, char **argv)
 49 {
 50     struct key *k, *kk;
 51     struct value *v, *found;
 52     struct hashtable *h;
 53     struct hashtable_itr *itr;
 54     int i;

 56     h = create_hashtable(16, hashfromkey, equalkeys);
 57     if (NULL == h) exit(-1); /*oom*/


 60 /*****************************************************************************/
```



```
 61 /* Insertion */
 62     for (i = 0; i < ITEM_COUNT; i++)
 63     {
 64         k = (struct key *)malloc(sizeof(struct key));
 65         if (NULL == k) {
 66             printf("ran out of memory allocating a key\n");
 67             return 1;
 68         }
 69         k->one_ip = 0xcfccee40 + i;
 70         k->two_ip = 0xcf0cee67 - (5 * i);
 71         k->one_port = 22 + (7 * i);
 72         k->two_port = 5522 - (3 * i);
 73
 74         v = (struct value *)malloc(sizeof(struct value));
 75         v->id = "a value";
 76
 77         if (!insert_some(h,k,v)) exit(-1); /*oom*/
 78     }
 79     printf("After insertion, hashtable contains %u items.\n",
 80             hashtable_count(h));

 82 /*****************************************************************************/
 83 /* Hashtable search */
 84     k = (struct key *)malloc(sizeof(struct key));
 85     if (NULL == k) {
 86         printf("ran out of memory allocating a key\n");
 87         return 1;
 88     }
 89
 90     for (i = 0; i < ITEM_COUNT; i++)
 91     {
 92         k->one_ip = 0xcfccee40 + i;
 93         k->two_ip = 0xcf0cee67 - (5 * i);
 94         k->one_port = 22 + (7 * i);
 95         k->two_port = 5522 - (3 * i);
 96
 97         if (NULL == (found = search_some(h,k))) {
 98             printf("BUG: key not found\n");
 99         }
100     }

102 /*****************************************************************************/
103 /* Hashtable iteration */
104     /* Iterator constructor only returns a valid iterator if
105      * the hashtable is not empty */
106     itr = hashtable_iterator(h);
107     i = 0;
108     if (hashtable_count(h) > 0)
109     {
110         do {
111             kk = hashtable_iterator_key(itr);
112             v = hashtable_iterator_value(itr);
113             /* here (kk,v) are a valid (key, value) pair */
114             /* We could call 'hashtable_remove(h,kk)' - and this operation
115              * 'free's kk. However, the iterator is then broken.
116              * This is why hashtable_iterator_remove exists - see below.
117              */
118             i++;

120         } while (hashtable_iterator_advance(itr));
121     }
122     printf("Iterated through %u entries.\n", i);

124 /*****************************************************************************/
125 /* Hashtable iterator search */
```

```
127     /* Try the search some method */
128     for (i = 0; i < ITEM_COUNT; i++)
129     {
130         k->one_ip = 0xcfccee40 + i;
131         k->two_ip = 0xcf0cee67 - (5 * i);
132         k->one_port = 22 + (7 * i);
133         k->two_port = 5522 - (3 * i);
134 
135         if (0 == search_itr_some(itr,h,k)) {
136             printf("BUG: key not found searching with iterator");
137         }
138     }
140 /*****************************************************************************/
141 /* Hashtable removal */
143     for (i = 0; i < ITEM_COUNT; i++)
144     {
145         k->one_ip = 0xcfccee40 + i;
146         k->two_ip = 0xcf0cee67 - (5 * i);
147         k->one_port = 22 + (7 * i);
148         k->two_port = 5522 - (3 * i);
149 
150         if (NULL == (found = remove_some(h,k))) {
151             printf("BUG: key not found for removal\n");
152         }
153     }
154     printf("After removal, hashtable contains %u items.\n",
155             hashtable_count(h));
157 /*****************************************************************************/
158 /* Hashtable destroy and create */
160     hashtable_destroy(h, 1);
161     h = NULL;
162     free(k);
164     h = create_hashtable(160, hashfromkey, equalkeys);
165     if (NULL == h) {
166         printf("out of memory allocating second hashtable\n");
167         return 1;
168     }
170 /*****************************************************************************/
171 /* Hashtable insertion */
173     for (i = 0; i < ITEM_COUNT; i++)
174     {
175         k = (struct key *)malloc(sizeof(struct key));
176         k->one_ip = 0xcfccee40 + i;
177         k->two_ip = 0xcf0cee67 - (5 * i);
178         k->one_port = 22 + (7 * i);
179         k->two_port = 5522 - (3 * i);
180 
181         v = (struct value *)malloc(sizeof(struct value));
182         v->id = "a value";
183 
184         if (!insert_some(h,k,v))
185         {
186             printf("out of memory inserting into second hashtable\n");
187             return 1;
188         }
189     }
190     printf("After insertion, hashtable contains %u items.\n",
191             hashtable_count(h));
```



```
193 /*****************************************************************************/
194 /* Hashtable iterator search and iterator remove */
196     k = (struct key *)malloc(sizeof(struct key));
197     if (NULL == k) {
198         printf("ran out of memory allocating a key\n");
199         return 1;
200     }
201 
202     for (i = ITEM_COUNT - 1; i >= 0; i = i - 7)
203     {
204         k->one_ip = 0xcfccee40 + i;
205         k->two_ip = 0xcf0cee67 - (5 * i);
206         k->one_port = 22 + (7 * i);
207         k->two_port = 5522 - (3 * i);
208 
209         if (0 == search_itr_some(itr, h, k)) {
210             printf("BUG: key %u not found for search preremoval using iterator\n
211             return 1;
212         }
213         if (0 == hashtable_iterator_remove(itr)) {
214             printf("BUG: key not found for removal using iterator\n");
215             return 1;
216         }
217     }
218     free(itr);
220 /*****************************************************************************/
221 /* Hashtable iterator remove and advance */
223     for (itr = hashtable_iterator(h);
224          hashtable_iterator_remove(itr) != 0; ) {
225         ;
226     }
227     free(itr);
228     printf("After removal, hashtable contains %u items.\n",
229             hashtable_count(h));
231 /*****************************************************************************/
232 /* Hashtable destroy */
234     hashtable_destroy(h, 1);
235     free(k);
236     return 0;
237 }
239 /*
240  * Copyright (c) 2002, 2004, Christopher Clark
241  * All rights reserved.
242  *
243  * Redistribution and use in source and binary forms, with or without
244  * modification, are permitted provided that the following conditions
245  * are met:
246  *
247  * * Redistributions of source code must retain the above copyright
248  * notice, this list of conditions and the following disclaimer.
249  *
250  * * Redistributions in binary form must reproduce the above copyright
251  * notice, this list of conditions and the following disclaimer in the
252  * documentation and/or other materials provided with the distribution.
253  *
254  * * Neither the name of the original author; nor the names of any contributors
255  * may be used to endorse or promote products derived from this software
256  * without specific prior written permission.
257  *
258  *
```

```
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER
 * OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
 * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
 * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
 * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

```
**********************************************************
   14535 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/dissect.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/dissect.c
 *
 * Started by Oleg Nesterov <oleg@redhat.com>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include "dissect.h"

#define U_VOID   0x00
#define U_SELF  ((1 << U_SHIFT) - 1)
#define U_MASK  (U_R_VAL | U_W_VAL | U_R_AOF)

#define DO_LIST(l__, p__, expr__)               \
        do {                                    \
                typeof(l__->list[0]) p__;       \
                FOR_EACH_PTR(l__, p__)          \
                        expr__;                 \
                END_FOR_EACH_PTR(p__);          \
        } while (0)

#define DO_2_LIST(l1__,l2__, p1__,p2__, expr__) \
        do {                                    \
                typeof(l1__->list[0]) p1__;     \
                typeof(l2__->list[0]) p2__;     \
                PREPARE_PTR_LIST(l1__, p1__);   \
                FOR_EACH_PTR(l2__, p2__)        \
                        expr__;                 \
                        NEXT_PTR_LIST(p1__);    \
                END_FOR_EACH_PTR(p2__);         \
                FINISH_PTR_LIST(p1__);          \
        } while (0)


typedef unsigned usage_t;

static struct reporter *reporter;
static struct symbol *return_type;

static void do_sym_list(struct symbol_list *list);

static struct symbol
        *base_type(struct symbol *sym),
```



```
        *do_initializer(struct symbol *type, struct expression *expr),
        *do_expression(usage_t mode, struct expression *expr),
        *do_statement(usage_t mode, struct statement *stmt);

static inline int is_ptr(struct symbol *type)
{
        return type->type == SYM_PTR || type->type == SYM_ARRAY;
}

static inline usage_t u_rval(usage_t mode)
{
        return mode & (U_R_VAL | (U_MASK << U_SHIFT))
                ? U_R_VAL : 0;
}

static inline usage_t u_addr(usage_t mode)
{
        return mode = mode & U_MASK
                ? U_R_AOF | (mode & U_W_AOF) : 0;
}

static usage_t u_lval(struct symbol *type)
{
        int wptr = is_ptr(type) && !(type->ctype.modifiers & MOD_CONST);
        return wptr || type == &bad_ctype
                ? U_W_AOF | U_R_VAL : U_R_VAL;
}

static usage_t fix_mode(struct symbol *type, usage_t mode)
{
        mode &= (U_SELF | (U_SELF << U_SHIFT));

        switch (type->type) {
                case SYM_BASETYPE:
                        if (!type->ctype.base_type)
                                break;
                case SYM_ENUM:
                case SYM_BITFIELD:
                        if (mode & U_MASK)
                                mode &= U_SELF;
                default:

                break; case SYM_FN:
                        if (mode & U_R_VAL)
                                mode |= U_R_AOF;
                        mode &= ~(U_R_VAL | U_W_AOF);

                break; case SYM_ARRAY:
                        if (mode & (U_MASK << U_SHIFT))
                                mode >>= U_SHIFT;
                        else if (mode != U_W_VAL)
                                mode = u_addr(mode);
        }

        if (!(mode & U_R_AOF))
                mode &= ~U_W_AOF;

        return mode;
}

static inline struct symbol *no_member(struct ident *name)
{
        static struct symbol sym = {
                .type = SYM_BAD,
        };
```

```
127         sym.ctype.base_type = &bad_ctype;
128         sym.ident = name;

130         return &sym;
131 }

133 static struct symbol *report_member(usage_t mode, struct position *pos,
134                                     struct symbol *type, struct symbol *mem)
135 {
136         struct symbol *ret = mem->ctype.base_type;

138         if (reporter->r_member)
139                 reporter->r_member(fix_mode(ret, mode), pos, type, mem);

141         return ret;
142 }

144 static void report_implicit(usage_t mode, struct position *pos, struct symbol *t
145 {
146         if (type->type != SYM_STRUCT && type->type != SYM_UNION)
147                 return;

149         if (!reporter->r_member)
150                 return;

152         if (type->ident != NULL)
153                 reporter->r_member(mode, pos, type, NULL);

155         DO_LIST(type->symbol_list, mem,
156                 report_implicit(mode, pos, base_type(mem)));
157 }

159 static inline struct symbol *expr_symbol(struct expression *expr)
160 {
161         struct symbol *sym = expr->symbol;

163         if (!sym) {
164                 sym = lookup_symbol(expr->symbol_name, NS_SYMBOL);

166                 if (!sym) {
167                         sym = alloc_symbol(expr->pos, SYM_BAD);
168                         bind_symbol(sym, expr->symbol_name, NS_SYMBOL);
169                         sym->ctype.modifiers = MOD_EXTERN;
170                 }
171         }

173         if (!sym->ctype.base_type)
174                 sym->ctype.base_type = &bad_ctype;

176         return sym;
177 }

179 static struct symbol *report_symbol(usage_t mode, struct expression *expr)
180 {
181         struct symbol *sym = expr_symbol(expr);
182         struct symbol *ret = base_type(sym);

184         if (0 && ret->type == SYM_ENUM)
185                 return report_member(mode, &expr->pos, ret, expr->symbol);

187         if (reporter->r_symbol)
188                 reporter->r_symbol(fix_mode(ret, mode), &expr->pos, sym);

190         return ret;
191 }
```



```
193 static inline struct ident *mk_name(struct ident *root, struct ident *node)
194 {
195         char name[256];

197         snprintf(name, sizeof(name), "%.*s:%.*s",
198                         root ? root->len : 0, root ? root->name : "",
199                         node ? node->len : 0, node ? node->name : "");

201         return built_in_ident(name);
202 }

204 static void examine_sym_node(struct symbol *node, struct ident *root)
205 {
206         struct symbol *base;
207         struct ident *name;

209         if (node->examined)
210                 return;

212         node->examined = 1;
213         name = node->ident;

215         while ((base = node->ctype.base_type) != NULL)
216                 switch (base->type) {
217                 case SYM_TYPEOF:
218                         node->ctype.base_type =
219                                 do_expression(U_VOID, base->initializer);
220                         break;

222                 case SYM_ARRAY:
223                         do_expression(U_R_VAL, base->array_size);
224                 case SYM_PTR: case SYM_FN:
225                         node = base;
226                         break;

228                 case SYM_STRUCT: case SYM_UNION: //case SYM_ENUM:
229                         if (base->evaluated)
230                                 return;
231                         if (!base->symbol_list)
232                                 return;
233                         base->evaluated = 1;

235                         if (!base->ident && name)
236                                 base->ident = mk_name(root, name);
237                         if (base->ident && reporter->r_symdef)
238                                 reporter->r_symdef(base);
239                         DO_LIST(base->symbol_list, mem,
240                                 examine_sym_node(mem, base->ident ?: root));
241                 default:
242                         return;
243                 }
244 }

246 static struct symbol *base_type(struct symbol *sym)
247 {
248         if (!sym)
249                 return &bad_ctype;

251         if (sym->type == SYM_NODE)
252                 examine_sym_node(sym, NULL);

254         return sym->ctype.base_type     // builtin_fn_type
255                 ?: &bad_ctype;
256 }

258 static struct symbol *__lookup_member(struct symbol *type, struct ident *name, i
```

```
{
        struct symbol *node;
        int addr = 0;

        FOR_EACH_PTR(type->symbol_list, node)
                if (!name) {
                        if (addr == *p_addr)
                                return node;
                }
                else if (node->ident == NULL) {
                        node = __lookup_member(node->ctype.base_type, name, NULL
                        if (node)
                                goto found;
                }
                else if (node->ident == name) {
found:
                        if (p_addr)
                                *p_addr = addr;
                        return node;
                }
                addr++;
        END_FOR_EACH_PTR(node);

        return NULL;
}

static struct symbol *lookup_member(struct symbol *type, struct ident *name, int
{
        return __lookup_member(type, name, addr)
                ?: no_member(name);
}

static struct expression *peek_preop(struct expression *expr, int op)
{
        do {
                if (expr->type != EXPR_PREOP)
                        break;
                if (expr->op == op)
                        return expr->unop;
                if (expr->op == '(')
                        expr = expr->unop;
                else
                        break;
        } while (expr);

        return NULL;
}

static struct symbol *do_expression(usage_t mode, struct expression *expr)
{
        struct symbol *ret = &int_ctype;

again:
        if (expr) switch (expr->type) {
        default:
                warning(expr->pos, "bad expr->type: %d", expr->type);

        case EXPR_TYPE:         // [struct T]; Why ???
        case EXPR_VALUE:
        case EXPR_FVALUE:

        break; case EXPR_LABEL:
                ret = &label_ctype;

        break; case EXPR_STRING:
                ret = &string_ctype;
```



```
        break; case EXPR_STATEMENT:
                ret = do_statement(mode, expr->statement);

        break; case EXPR_SIZEOF: case EXPR_ALIGNOF: case EXPR_PTRSIZEOF:
                do_expression(U_VOID, expr->cast_expression);

        break; case EXPR_COMMA:
                do_expression(U_VOID, expr->left);
                ret = do_expression(mode, expr->right);

        break; case EXPR_CAST: case EXPR_FORCE_CAST: //case EXPR_IMPLIED_CAST:
                ret = base_type(expr->cast_type);
                do_initializer(ret, expr->cast_expression);

        break; case EXPR_COMPARE: case EXPR_LOGICAL:
                mode = u_rval(mode);
                do_expression(mode, expr->left);
                do_expression(mode, expr->right);

        break; case EXPR_CONDITIONAL: //case EXPR_SELECT:
                do_expression(expr->cond_true
                                        ? U_R_VAL : U_R_VAL | mode,
                              expr->conditional);
                ret = do_expression(mode, expr->cond_true);
                ret = do_expression(mode, expr->cond_false);

        break; case EXPR_CALL:
                ret = do_expression(U_R_PTR, expr->fn);
                if (is_ptr(ret))
                        ret = ret->ctype.base_type;
                DO_2_LIST(ret->arguments, expr->args, arg, val,
                        do_expression(u_lval(base_type(arg)), val));
                ret = ret->type == SYM_FN ? base_type(ret)
                        : &bad_ctype;

        break; case EXPR_ASSIGNMENT:
                mode |= U_W_VAL | U_R_VAL;
                if (expr->op == '=')
                        mode &= ~U_R_VAL;
                ret = do_expression(mode, expr->left);
                report_implicit(mode, &expr->pos, ret);
                mode = expr->op == '='
                        ? u_lval(ret) : U_R_VAL;
                do_expression(mode, expr->right);

        break; case EXPR_BINOP: {
                struct symbol *l, *r;
                mode |= u_rval(mode);
                l = do_expression(mode, expr->left);
                r = do_expression(mode, expr->right);
                if (expr->op != '+' && expr->op != '-')
                        ;
                else if (!is_ptr_type(r))
                        ret = l;
                else if (!is_ptr_type(l))
                        ret = r;
        }

        break; case EXPR_PREOP: case EXPR_POSTOP: {
                struct expression *unop = expr->unop;

                switch (expr->op) {
                case SPECIAL_INCREMENT:
                case SPECIAL_DECREMENT:
                        mode |= U_W_VAL | U_R_VAL;
```

```
391                 default:
392                         mode |= u_rval(mode);
393                 case '(':
394                         ret = do_expression(mode, unop);

396                 break; case '&':
397                         if ((expr = peek_preop(unop, '*')))
398                                 goto again;
399                         ret = alloc_symbol(unop->pos, SYM_PTR);
400                         ret->ctype.base_type =
401                                 do_expression(u_addr(mode), unop);

403                 break; case '*':
404                         if ((expr = peek_preop(unop, '&')))
405                                 goto again;
406                         if (mode & (U_MASK << U_SHIFT))
407                                 mode |= U_R_VAL;
408                         mode <<= U_SHIFT;
409                         if (mode & (U_R_AOF << U_SHIFT))
410                                 mode |= U_R_VAL;
411                         if (mode & (U_W_VAL << U_SHIFT))
412                                 mode |= U_W_AOF;
413                         ret = do_expression(mode, unop);
414                         ret = is_ptr(ret) ? base_type(ret)
415                                 : &bad_ctype;
416                 }
417         }

419         break; case EXPR_DEREF: {
420                 struct symbol *p_type;
421                 usage_t p_mode;

423                 p_mode = mode & U_SELF;
424                 if (!(mode & U_MASK) && (mode & (U_MASK << U_SHIFT)))
425                         p_mode = U_R_VAL;
426                 p_type = do_expression(p_mode, expr->deref);

428                 ret = report_member(mode, &expr->pos, p_type,
429                         lookup_member(p_type, expr->member, NULL));
430         }

432         break; case EXPR_OFFSETOF: {
433                 struct symbol *in = base_type(expr->in);

435                 do {
436                         if (expr->op == '.') {
437                                 in = report_member(U_VOID, &expr->pos, in,
438                                         lookup_member(in, expr->ident, NULL));
439                         } else {
440                                 do_expression(U_R_VAL, expr->index);
441                                 in = in->ctype.base_type;
442                         }
443                 } while ((expr = expr->down));
444         }

446         break; case EXPR_SYMBOL:
447                 ret = report_symbol(mode, expr);
448         }

450         return ret;
451 }

453 static void do_asm_xputs(usage_t mode, struct expression_list *xputs)
454 {
455         int nr = 0;
```



```
457         DO_LIST(xputs, expr,
458                 if (++nr % 3 == 0)
459                         do_expression(U_W_AOF | mode, expr));
460 }

462 static struct symbol *do_statement(usage_t mode, struct statement *stmt)
463 {
464         struct symbol *ret = &void_ctype;

466         if (stmt) switch (stmt->type) {
467         default:
468                 warning(stmt->pos, "bad stmt->type: %d", stmt->type);

470         case STMT_NONE:
471         case STMT_RANGE:
472         case STMT_CONTEXT:

474         break; case STMT_DECLARATION:
475                 do_sym_list(stmt->declaration);

477         break; case STMT_EXPRESSION:
478                 ret = do_expression(mode, stmt->expression);

480         break; case STMT_RETURN:
481                 do_expression(u_lval(return_type), stmt->expression);

483         break; case STMT_ASM:
484                 do_expression(U_R_VAL, stmt->asm_string);
485                 do_asm_xputs(U_W_VAL, stmt->asm_outputs);
486                 do_asm_xputs(U_R_VAL, stmt->asm_inputs);

488         break; case STMT_COMPOUND: {
489                 int count;

491                 count = statement_list_size(stmt->stmts);
492                 DO_LIST(stmt->stmts, st,
493                         ret = do_statement(--count ? U_VOID : mode, st));
494         }

496         break; case STMT_ITERATOR:
497                 do_sym_list(stmt->iterator_syms);
498                 do_statement(U_VOID, stmt->iterator_pre_statement);
499                 do_expression(U_R_VAL, stmt->iterator_pre_condition);
500                 do_statement(U_VOID, stmt->iterator_post_statement);
501                 do_statement(U_VOID, stmt->iterator_statement);
502                 do_expression(U_R_VAL, stmt->iterator_post_condition);

504         break; case STMT_IF:
505                 do_expression(U_R_VAL, stmt->if_conditional);
506                 do_statement(U_VOID, stmt->if_true);
507                 do_statement(U_VOID, stmt->if_false);

509         break; case STMT_SWITCH:
510                 do_expression(U_R_VAL, stmt->switch_expression);
511                 do_statement(U_VOID, stmt->switch_statement);

513         break; case STMT_CASE:
514                 do_expression(U_R_VAL, stmt->case_expression);
515                 do_expression(U_R_VAL, stmt->case_to);
516                 do_statement(U_VOID, stmt->case_statement);

518         break; case STMT_GOTO:
519                 do_expression(U_R_PTR, stmt->goto_expression);

521         break; case STMT_LABEL:
522                 do_statement(mode, stmt->label_statement);
```

```
524         }
526         return ret;
527 }
529 static struct symbol *do_initializer(struct symbol *type, struct expression *exp
530 {
531         struct symbol *m_type;
532         struct expression *m_expr;
533         int m_addr;
535         if (expr) switch (expr->type) {
536         default:
537                 do_expression(u_lval(type), expr);
539         break; case EXPR_INDEX:
540                 do_initializer(base_type(type), expr->idx_expression);
542         break; case EXPR_INITIALIZER:
543                 m_addr = 0;
544                 FOR_EACH_PTR(expr->expr_list, m_expr) {
545                         if (type->type == SYM_ARRAY) {
546                                 m_type = base_type(type);
547                                 if (m_expr->type == EXPR_INDEX)
548                                         m_expr = m_expr->idx_expression;
549                         } else {
550                                 int *m_atop = &m_addr;
552                                 m_type = type;
553                                 while (m_expr->type == EXPR_IDENTIFIER) {
554                                         m_type = report_member(U_W_VAL, &m_expr-
555                                                                lookup_member(m_type, m_
556                                         m_expr = m_expr->ident_expression;
557                                         m_atop = NULL;
558                                 }
560                                 if (m_atop) {
561                                         m_type = report_member(U_W_VAL, &m_expr-
562                                                                lookup_member(m_type, NU
563                                 }
565                                 if (m_expr->type != EXPR_INITIALIZER)
566                                         report_implicit(U_W_VAL, &m_expr->pos, m
567                         }
568                         do_initializer(m_type, m_expr);
569                         m_addr++;
570                 } END_FOR_EACH_PTR(m_expr);
571         }
573         return type;
574 }
576 static inline struct symbol *do_symbol(struct symbol *sym)
577 {
578         struct symbol *type;
580         type = base_type(sym);
582         if (reporter->r_symdef)
583                 reporter->r_symdef(sym);
585         switch (type->type) {
586         default:
587                 if (!sym->initializer)
588                         break;
```



```
589                 if (reporter->r_symbol)
590                         reporter->r_symbol(U_W_VAL, &sym->pos, sym);
591                 do_initializer(type, sym->initializer);
593         break; case SYM_FN:
594                 do_sym_list(type->arguments);
595                 return_type = base_type(type);
596                 do_statement(U_VOID, sym->ctype.modifiers & MOD_INLINE
597                                         ? type->inline_stmt
598                                         : type->stmt);
599         }
601         return type;
602 }
604 static void do_sym_list(struct symbol_list *list)
605 {
606         DO_LIST(list, sym, do_symbol(sym));
607 }
609 void dissect(struct symbol_list *list, struct reporter *rep)
610 {
611         reporter = rep;
612         do_sym_list(list);
613 }
```

```
**********************************************************
     563 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/dissect.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef DISSECT_H
#define DISSECT_H

#include <stdio.h>
#include "parse.h"
#include "expression.h"

#define U_SHIFT         8

#define U_R_AOF         0x01
#define U_W_AOF         0x02

#define U_R_VAL         0x04
#define U_W_VAL         0x08

#define U_R_PTR         (U_R_VAL << U_SHIFT)
#define U_W_PTR         (U_W_VAL << U_SHIFT)

struct reporter
{
        void (*r_symdef)(struct symbol *);

        void (*r_symbol)(unsigned, struct position *, struct symbol *);
        void (*r_member)(unsigned, struct position *, struct symbol *, struct sy
};

extern void dissect(struct symbol_list *, struct reporter *);

#endif
```

```
**********************************************************
   92246 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/evaluate.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/evaluate.c
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * Evaluate constant expressions.
 */
#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <limits.h>

#include "lib.h"
#include "allocate.h"
#include "parse.h"
#include "token.h"
#include "symbol.h"
#include "target.h"
#include "expression.h"

struct symbol *current_fn;

static struct symbol *degenerate(struct expression *expr);
static struct symbol *evaluate_symbol(struct symbol *sym);

static struct symbol *evaluate_symbol_expression(struct expression *expr)
{
        struct expression *addr;
        struct symbol *sym = expr->symbol;
        struct symbol *base_type;

        if (!sym) {
                expression_error(expr, "undefined identifier '%s'", show_ident(e
                return NULL;
        }
```



```
        examine_symbol_type(sym);

        base_type = get_base_type(sym);
        if (!base_type) {
                expression_error(expr, "identifier '%s' has no type", show_ident
                return NULL;
        }

        addr = alloc_expression(expr->pos, EXPR_SYMBOL);
        addr->symbol = sym;
        addr->symbol_name = expr->symbol_name;
        addr->ctype = &lazy_ptr_ctype;  /* Lazy evaluation: we need to do a prop
        addr->flags = expr->flags;
        expr->type = EXPR_PREOP;
        expr->op = '*';
        expr->unop = addr;
        expr->flags = CEF_NONE;

        /* The type of a symbol is the symbol itself! */
        expr->ctype = sym;
        return sym;
}

static struct symbol *evaluate_string(struct expression *expr)
{
        struct symbol *sym = alloc_symbol(expr->pos, SYM_NODE);
        struct symbol *array = alloc_symbol(expr->pos, SYM_ARRAY);
        struct expression *addr = alloc_expression(expr->pos, EXPR_SYMBOL);
        struct expression *initstr = alloc_expression(expr->pos, EXPR_STRING);
        unsigned int length = expr->string->length;

        sym->array_size = alloc_const_expression(expr->pos, length);
        sym->bit_size = bytes_to_bits(length);
        sym->ctype.alignment = 1;
        sym->string = 1;
        sym->ctype.modifiers = MOD_STATIC;
        sym->ctype.base_type = array;
        sym->initializer = initstr;

        initstr->ctype = sym;
        initstr->string = expr->string;

        array->array_size = sym->array_size;
        array->bit_size = bytes_to_bits(length);
        array->ctype.alignment = 1;
        array->ctype.modifiers = MOD_STATIC;
        array->ctype.base_type = &char_ctype;

        addr->symbol = sym;
        addr->ctype = &lazy_ptr_ctype;
        addr->flags = CEF_ADDR;

        expr->type = EXPR_PREOP;
        expr->op = '*';
        expr->unop = addr;
        expr->ctype = sym;
        return sym;
}

/* type has come from classify_type and is an integer type */
static inline struct symbol *integer_promotion(struct symbol *type)
{
        unsigned long mod =  type->ctype.modifiers;
        int width = type->bit_size;

        /*
```

```
127          * Bitfields always promote to the base type,
128          * even if the bitfield might be bigger than
129          * an "int".
130          */
131         if (type->type == SYM_BITFIELD) {
132                 type = type->ctype.base_type;
133         }
134         mod = type->ctype.modifiers;
135         if (width < bits_in_int)
136                 return &int_ctype;

138         /* If char/short has as many bits as int, it still gets "promoted" */
139         if (mod & (MOD_CHAR | MOD_SHORT)) {
140                 if (mod & MOD_UNSIGNED)
141                         return &uint_ctype;
142                 return &int_ctype;
143         }
144         return type;
145 }

147 /*
148  * integer part of usual arithmetic conversions:
149  *      integer promotions are applied
150  *      if left and right are identical, we are done
151  *      if signedness is the same, convert one with lower rank
152  *      unless unsigned argument has rank lower than signed one, convert the
153  *      signed one.
154  *      if signed argument is bigger than unsigned one, convert the unsigned.
155  *      otherwise, convert signed.
156  *
157  * Leaving aside the integer promotions, that is equivalent to
158  *      if identical, don't convert
159  *      if left is bigger than right, convert right
160  *      if right is bigger than left, convert right
161  *      otherwise, if signedness is the same, convert one with lower rank
162  *      otherwise convert the signed one.
163  */
164 static struct symbol *bigger_int_type(struct symbol *left, struct symbol *right)
165 {
166         unsigned long lmod, rmod;

168         left = integer_promotion(left);
169         right = integer_promotion(right);

171         if (left == right)
172                 goto left;

174         if (left->bit_size > right->bit_size)
175                 goto left;

177         if (right->bit_size > left->bit_size)
178                 goto right;

180         lmod = left->ctype.modifiers;
181         rmod = right->ctype.modifiers;
182         if ((lmod ^ rmod) & MOD_UNSIGNED) {
183                 if (lmod & MOD_UNSIGNED)
184                         goto left;
185         } else if ((lmod & ~rmod) & (MOD_LONG_ALL))
186                 goto left;
187 right:
188         left = right;
189 left:
190         return left;
191 }
```



```
193 static int same_cast_type(struct symbol *orig, struct symbol *new)
194 {
195         return orig->bit_size == new->bit_size &&
196                orig->bit_offset == new->bit_offset;
197 }

199 static struct symbol *base_type(struct symbol *node, unsigned long *modp, unsign
200 {
201         unsigned long mod, as;

203         mod = 0; as = 0;
204         while (node) {
205                 mod |= node->ctype.modifiers;
206                 as |= node->ctype.as;
207                 if (node->type == SYM_NODE) {
208                         node = node->ctype.base_type;
209                         continue;
210                 }
211                 break;
212         }
213         *modp = mod & ~MOD_IGNORE;
214         *asp = as;
215         return node;
216 }

218 static int is_same_type(struct expression *expr, struct symbol *new)
219 {
220         struct symbol *old = expr->ctype;
221         unsigned long oldmod, newmod, oldas, newas;

223         old = base_type(old, &oldmod, &oldas);
224         new = base_type(new, &newmod, &newas);

226         /* Same base type, same address space? */
227         if (old == new && oldas == newas) {
228                 unsigned long difmod;

230                 /* Check the modifier bits. */
231                 difmod = (oldmod ^ newmod) & ~MOD_NOCAST;

233                 /* Exact same type? */
234                 if (!difmod)
235                         return 1;

237                 /*
238                  * Not the same type, but differs only in "const".
239                  * Don't warn about MOD_NOCAST.
240                  */
241                 if (difmod == MOD_CONST)
242                         return 0;
243         }
244         if ((oldmod | newmod) & MOD_NOCAST) {
245                 const char *tofrom = "to/from";
246                 if (!(newmod & MOD_NOCAST))
247                         tofrom = "from";
248                 if (!(oldmod & MOD_NOCAST))
249                         tofrom = "to";
250                 warning(expr->pos, "implicit cast %s nocast type", tofrom);
251         }
252         return 0;
253 }

255 static void
256 warn_for_different_enum_types (struct position pos,
257                                struct symbol *typea,
258                                struct symbol *typeb)
```

```
{
        if (!Wenum_mismatch)
                return;
        if (typea->type == SYM_NODE)
                typea = typea->ctype.base_type;
        if (typeb->type == SYM_NODE)
                typeb = typeb->ctype.base_type;

        if (typea == typeb)
                return;

        if (typea->type == SYM_ENUM && typeb->type == SYM_ENUM) {
                warning(pos, "mixing different enum types");
                info(pos, "    %s versus", show_typename(typea));
                info(pos, "    %s", show_typename(typeb));
        }
}

static int cast_flags(struct expression *expr, struct expression *target);
static struct symbol *cast_to_bool(struct expression *expr);

/*
 * This gets called for implicit casts in assignments and
 * integer promotion. We often want to try to move the
 * cast down, because the ops involved may have been
 * implicitly cast up, and we can get rid of the casts
 * early.
 */
static struct expression * cast_to(struct expression *old, struct symbol *type)
{
        struct expression *expr;

        warn_for_different_enum_types (old->pos, old->ctype, type);

        if (old->ctype != &null_ctype && is_same_type(old, type))
                return old;

        /*
         * See if we can simplify the op. Move the cast down.
         */
        switch (old->type) {
        case EXPR_PREOP:
                if (old->ctype->bit_size < type->bit_size)
                        break;
                if (old->op == '~') {
                        old->ctype = type;
                        old->unop = cast_to(old->unop, type);
                        return old;
                }
                break;

        case EXPR_IMPLIED_CAST:
                warn_for_different_enum_types(old->pos, old->ctype, type);

                if (old->ctype->bit_size >= type->bit_size) {
                        struct expression *orig = old->cast_expression;
                        if (same_cast_type(orig->ctype, type))
                                return orig;
                        if (old->ctype->bit_offset == type->bit_offset) {
                                old->ctype = type;
                                old->cast_type = type;
                                return old;
                        }
                }
                break;
```



```
        default:
                /* nothing */;
        }

        expr = alloc_expression(old->pos, EXPR_IMPLIED_CAST);
        expr->ctype = type;
        expr->cast_type = type;
        expr->cast_expression = old;
        expr->flags = cast_flags(expr, old);

        if (is_bool_type(type))
                cast_to_bool(expr);

        return expr;
}

enum {
        TYPE_NUM = 1,
        TYPE_BITFIELD = 2,
        TYPE_RESTRICT = 4,
        TYPE_FLOAT = 8,
        TYPE_PTR = 16,
        TYPE_COMPOUND = 32,
        TYPE_FOULED = 64,
        TYPE_FN = 128,
};

static inline int classify_type(struct symbol *type, struct symbol **base)
{
        static int type_class[SYM_BAD + 1] = {
                [SYM_PTR] = TYPE_PTR,
                [SYM_FN] = TYPE_PTR | TYPE_FN,
                [SYM_ARRAY] = TYPE_PTR | TYPE_COMPOUND,
                [SYM_STRUCT] = TYPE_COMPOUND,
                [SYM_UNION] = TYPE_COMPOUND,
                [SYM_BITFIELD] = TYPE_NUM | TYPE_BITFIELD,
                [SYM_RESTRICT] = TYPE_NUM | TYPE_RESTRICT,
                [SYM_FOULED] = TYPE_NUM | TYPE_RESTRICT | TYPE_FOULED,
        };
        if (type->type == SYM_NODE)
                type = type->ctype.base_type;
        if (type->type == SYM_TYPEOF) {
                type = evaluate_expression(type->initializer);
                if (!type)
                        type = &bad_ctype;
                else if (type->type == SYM_NODE)
                        type = type->ctype.base_type;
        }
        if (type->type == SYM_ENUM)
                type = type->ctype.base_type;
        *base = type;
        if (type->type == SYM_BASETYPE) {
                if (type->ctype.base_type == &int_type)
                        return TYPE_NUM;
                if (type->ctype.base_type == &fp_type)
                        return TYPE_NUM | TYPE_FLOAT;
        }
        return type_class[type->type];
}

#define is_int(class) ((class & (TYPE_NUM | TYPE_FLOAT)) == TYPE_NUM)

static inline int is_string_type(struct symbol *type)
{
        if (type->type == SYM_NODE)
                type = type->ctype.base_type;
```

```
391         return type->type == SYM_ARRAY && is_byte_type(type->ctype.base_type);
392 }

394 static struct symbol *bad_expr_type(struct expression *expr)
395 {
396         sparse_error(expr->pos, "incompatible types for operation (%s)", show_sp
397         switch (expr->type) {
398         case EXPR_BINOP:
399         case EXPR_COMPARE:
400                 info(expr->pos, "   left side has type %s", show_typename(expr->
401                 info(expr->pos, "   right side has type %s", show_typename(expr-
402                 break;
403         case EXPR_PREOP:
404         case EXPR_POSTOP:
405                 info(expr->pos, "   argument has type %s", show_typename(expr->u
406                 break;
407         default:
408                 break;
409         }

411         expr->flags = CEF_NONE;
412         return expr->ctype = &bad_ctype;
413 }

415 static int restricted_value(struct expression *v, struct symbol *type)
416 {
417         if (v->type != EXPR_VALUE)
418                 return 1;
419         if (v->value != 0)
420                 return 1;
421         return 0;
422 }

424 static int restricted_binop(int op, struct symbol *type)
425 {
426         switch (op) {
427                 case '&':
428                 case '=':
429                 case SPECIAL_AND_ASSIGN:
430                 case SPECIAL_OR_ASSIGN:
431                 case SPECIAL_XOR_ASSIGN:
432                         return 1;       /* unfoul */
433                 case '|':
434                 case '^':
435                 case '?':
436                         return 2;       /* keep fouled */
437                 case SPECIAL_EQUAL:
438                 case SPECIAL_NOTEQUAL:
439                         return 3;       /* warn if fouled */
440                 default:
441                         return 0;       /* warn */
442         }
443 }

445 static int restricted_unop(int op, struct symbol **type)
446 {
447         if (op == '~') {
448                 if ((*type)->bit_size < bits_in_int)
449                         *type = befoul(*type);
450                 return 0;
451         } if (op == '+')
452                 return 0;
453         return 1;
454 }

456 /* type should be SYM_FOULED */
```



```
457 static inline struct symbol *unfoul(struct symbol *type)
458 {
459         return type->ctype.base_type;
460 }

462 static struct symbol *restricted_binop_type(int op,
463                                             struct expression *left,
464                                             struct expression *right,
465                                             int lclass, int rclass,
466                                             struct symbol *ltype,
467                                             struct symbol *rtype)
468 {
469         struct symbol *ctype = NULL;
470         if (lclass & TYPE_RESTRICT) {
471                 if (rclass & TYPE_RESTRICT) {
472                         if (ltype == rtype) {
473                                 ctype = ltype;
474                         } else if (lclass & TYPE_FOULED) {
475                                 if (unfoul(ltype) == rtype)
476                                         ctype = ltype;
477                         } else if (rclass & TYPE_FOULED) {
478                                 if (unfoul(rtype) == ltype)
479                                         ctype = rtype;
480                         }
481                 } else {
482                         if (!restricted_value(right, ltype))
483                                 ctype = ltype;
484                 }
485         } else if (!restricted_value(left, rtype))
486                 ctype = rtype;

488         if (ctype) {
489                 switch (restricted_binop(op, ctype)) {
490                 case 1:
491                         if ((lclass ^ rclass) & TYPE_FOULED)
492                                 ctype = unfoul(ctype);
493                         break;
494                 case 3:
495                         if (!(lclass & rclass & TYPE_FOULED))
496                                 break;
497                 case 0:
498                         ctype = NULL;
499                 default:
500                         break;
501                 }
502         }

504         return ctype;
505 }

507 static inline void unrestrict(struct expression *expr,
508                               int class, struct symbol **ctype)
509 {
510         if (class & TYPE_RESTRICT) {
511                 if (class & TYPE_FOULED)
512                         *ctype = unfoul(*ctype);
513                 warning(expr->pos, "%s degrades to integer",
514                         show_typename(*ctype));
515                 *ctype = (*ctype)->ctype.base_type; /* get to arithmetic type */
516         }
517 }

519 static struct symbol *usual_conversions(int op,
520                                         struct expression *left,
521                                         struct expression *right,
522                                         int lclass, int rclass,
```

```
523                                     struct symbol *ltype,
524                                     struct symbol *rtype)
525 {
526         struct symbol *ctype;

528         warn_for_different_enum_types(right->pos, left->ctype, right->ctype);

530         if ((lclass | rclass) & TYPE_RESTRICT)
531                 goto Restr;

533 Normal:
534         if (!(lclass & TYPE_FLOAT)) {
535                 if (!(rclass & TYPE_FLOAT))
536                         return bigger_int_type(ltype, rtype);
537                 else
538                         return rtype;
539         } else if (rclass & TYPE_FLOAT) {
540                 unsigned long lmod = ltype->ctype.modifiers;
541                 unsigned long rmod = rtype->ctype.modifiers;
542                 if (rmod & ~lmod & (MOD_LONG_ALL))
543                         return rtype;
544                 else
545                         return ltype;
546         } else
547                 return ltype;

549 Restr:
550         ctype = restricted_binop_type(op, left, right,
551                                       lclass, rclass, ltype, rtype);
552         if (ctype)
553                 return ctype;

555         unrestrict(left, lclass, &ltype);
556         unrestrict(right, rclass, &rtype);

558         goto Normal;
559 }

561 static inline int lvalue_expression(struct expression *expr)
562 {
563         return expr->type == EXPR_PREOP && expr->op == '*';
564 }

566 static struct symbol *evaluate_ptr_add(struct expression *expr, struct symbol *i
567 {
568         struct expression *index = expr->right;
569         struct symbol *ctype, *base;
570         int multiply;

572         classify_type(degenerate(expr->left), &ctype);
573         base = examine_pointer_target(ctype);

575         /*
576          * An address constant +/- an integer constant expression
577          * yields an address constant again [6.6(7)].
578          */
579         if ((expr->left->flags & CEF_ADDR) && (expr->right->flags & CEF_ICE))
580                 expr->flags = CEF_ADDR;

582         if (!base) {
583                 expression_error(expr, "missing type information");
584                 return NULL;
585         }
586         if (is_function(base)) {
587                 expression_error(expr, "arithmetics on pointers to functions");
588                 return NULL;
```



```
589         }

591         /* Get the size of whatever the pointer points to */
592         multiply = is_void_type(base) ? 1 : bits_to_bytes(base->bit_size);

594         if (ctype == &null_ctype)
595                 ctype = &ptr_ctype;
596         expr->ctype = ctype;

598         if (multiply == 1 && itype->bit_size >= bits_in_pointer)
599                 return ctype;

601         if (index->type == EXPR_VALUE) {
602                 struct expression *val = alloc_expression(expr->pos, EXPR_VALUE)
603                 unsigned long long v = index->value, mask;
604                 mask = 1ULL << (itype->bit_size - 1);
605                 if (v & mask)
606                         v |= -mask;
607                 else
608                         v &= mask - 1;
609                 v *= multiply;
610                 mask = 1ULL << (bits_in_pointer - 1);
611                 v &= mask | (mask - 1);
612                 val->value = v;
613                 val->ctype = ssize_t_ctype;
614                 expr->right = val;
615                 return ctype;
616         }

618         if (itype->bit_size < bits_in_pointer)
619                 index = cast_to(index, ssize_t_ctype);

621         if (multiply > 1) {
622                 struct expression *val = alloc_expression(expr->pos, EXPR_VALUE)
623                 struct expression *mul = alloc_expression(expr->pos, EXPR_BINOP)

625                 val->ctype = ssize_t_ctype;
626                 val->value = multiply;

628                 mul->op = '*';
629                 mul->ctype = ssize_t_ctype;
630                 mul->left = index;
631                 mul->right = val;
632                 index = mul;
633         }

635         expr->right = index;
636         return ctype;
637 }

639 static void examine_fn_arguments(struct symbol *fn);

641 #define MOD_IGN (MOD_VOLATILE | MOD_CONST | MOD_PURE)

643 const char *type_difference(struct ctype *c1, struct ctype *c2,
644         unsigned long mod1, unsigned long mod2)
645 {
646         unsigned long as1 = c1->as, as2 = c2->as;
647         struct symbol *t1 = c1->base_type;
648         struct symbol *t2 = c2->base_type;
649         int move1 = 1, move2 = 1;
650         mod1 |= c1->modifiers;
651         mod2 |= c2->modifiers;
652         for (;;) {
653                 unsigned long diff;
654                 int type;
```

```
655                 struct symbol *base1 = t1->ctype.base_type;
656                 struct symbol *base2 = t2->ctype.base_type;

658                 /*
659                  * FIXME! Collect alignment and context too here!
660                  */
661                 if (move1) {
662                         if (t1 && t1->type != SYM_PTR) {
663                                 mod1 |= t1->ctype.modifiers;
664                                 as1 |= t1->ctype.as;
665                         }
666                         move1 = 0;
667                 }

669                 if (move2) {
670                         if (t2 && t2->type != SYM_PTR) {
671                                 mod2 |= t2->ctype.modifiers;
672                                 as2 |= t2->ctype.as;
673                         }
674                         move2 = 0;
675                 }

677                 if (t1 == t2)
678                         break;
679                 if (!t1 || !t2)
680                         return "different types";

682                 if (t1->type == SYM_NODE || t1->type == SYM_ENUM) {
683                         t1 = base1;
684                         move1 = 1;
685                         if (!t1)
686                                 return "bad types";
687                         continue;
688                 }

690                 if (t2->type == SYM_NODE || t2->type == SYM_ENUM) {
691                         t2 = base2;
692                         move2 = 1;
693                         if (!t2)
694                                 return "bad types";
695                         continue;
696                 }

698                 move1 = move2 = 1;
699                 type = t1->type;
700                 if (type != t2->type)
701                         return "different base types";

703                 switch (type) {
704                 default:
705                         sparse_error(t1->pos,
706                                      "internal error: bad type in derived(%d)",
707                                      type);
708                         return "bad types";
709                 case SYM_RESTRICT:
710                         return "different base types";
711                 case SYM_UNION:
712                 case SYM_STRUCT:
713                         /* allow definition of incomplete structs and unions */
714                         if (t1->ident == t2->ident)
715                           return NULL;
716                         return "different base types";
717                 case SYM_ARRAY:
718                         /* XXX: we ought to compare sizes */
719                         break;
720                 case SYM_PTR:
```



```
721                         if (as1 != as2)
722                                 return "different address spaces";
723                         /* MOD_SPECIFIER is due to idiocy in parse.c */
724                         if ((mod1 ^ mod2) & ~MOD_IGNORE & ~MOD_SPECIFIER)
725                                 return "different modifiers";
726                         /* we could be lazier here */
727                         base1 = examine_pointer_target(t1);
728                         base2 = examine_pointer_target(t2);
729                         mod1 = t1->ctype.modifiers;
730                         as1 = t1->ctype.as;
731                         mod2 = t2->ctype.modifiers;
732                         as2 = t2->ctype.as;
733                         break;
734                 case SYM_FN: {
735                         struct symbol *arg1, *arg2;
736                         int i;

738                         if (as1 != as2)
739                                 return "different address spaces";
740                         if ((mod1 ^ mod2) & ~MOD_IGNORE & ~MOD_SIGNEDNESS)
741                                 return "different modifiers";
742                         mod1 = t1->ctype.modifiers;
743                         as1 = t1->ctype.as;
744                         mod2 = t2->ctype.modifiers;
745                         as2 = t2->ctype.as;

747                         if (t1->variadic != t2->variadic)
748                                 return "incompatible variadic arguments";
749                         examine_fn_arguments(t1);
750                         examine_fn_arguments(t2);
751                         PREPARE_PTR_LIST(t1->arguments, arg1);
752                         PREPARE_PTR_LIST(t2->arguments, arg2);
753                         i = 1;
754                         for (;;) {
755                                 const char *diffstr;
756                                 if (!arg1 && !arg2)
757                                         break;
758                                 if (!arg1 || !arg2)
759                                         return "different argument counts";
760                                 diffstr = type_difference(&arg1->ctype,
761                                                           &arg2->ctype,
762                                                           MOD_IGN, MOD_IGN);
763                                 if (diffstr) {
764                                         static char argdiff[80];
765                                         sprintf(argdiff, "incompatible argument
766                                         return argdiff;
767                                 }
768                                 NEXT_PTR_LIST(arg1);
769                                 NEXT_PTR_LIST(arg2);
770                                 i++;
771                         }
772                         FINISH_PTR_LIST(arg2);
773                         FINISH_PTR_LIST(arg1);
774                         break;
775                 }
776                 case SYM_BASETYPE:
777                         if (as1 != as2)
778                                 return "different address spaces";
779                         if (base1 != base2)
780                                 return "different base types";
781                         diff = (mod1 ^ mod2) & ~MOD_IGNORE;
782                         if (!diff)
783                                 return NULL;
784                         if (diff & MOD_SIZE)
785                                 return "different type sizes";
786                         else if (diff & ~MOD_SIGNEDNESS)
```

```
787                                 return "different modifiers";
788                         else
789                                 return "different signedness";
790                 }
791                 t1 = base1;
792                 t2 = base2;
793         }
794         if (as1 != as2)
795                 return "different address spaces";
796         if ((mod1 ^ mod2) & ~MOD_IGNORE & ~MOD_SIGNEDNESS)
797                 return "different modifiers";
798         return NULL;
799 }

801 static void bad_null(struct expression *expr)
802 {
803         if (Wnon_pointer_null)
804                 warning(expr->pos, "Using plain integer as NULL pointer");
805 }

807 static unsigned long target_qualifiers(struct symbol *type)
808 {
809         unsigned long mod = type->ctype.modifiers & MOD_IGN;
810         if (type->ctype.base_type && type->ctype.base_type->type == SYM_ARRAY)
811                 mod = 0;
812         return mod;
813 }

815 static struct symbol *evaluate_ptr_sub(struct expression *expr)
816 {
817         const char *typediff;
818         struct symbol *ltype, *rtype;
819         struct expression *l = expr->left;
820         struct expression *r = expr->right;
821         struct symbol *lbase;

823         classify_type(degenerate(l), &ltype);
824         classify_type(degenerate(r), &rtype);

826         lbase = examine_pointer_target(ltype);
827         examine_pointer_target(rtype);
828         typediff = type_difference(&ltype->ctype, &rtype->ctype,
829                                    target_qualifiers(rtype),
830                                    target_qualifiers(ltype));
831         if (typediff)
832                 expression_error(expr, "subtraction of different types can't wor

834         if (is_function(lbase)) {
835                 expression_error(expr, "subtraction of functions? Share your dru
836                 return NULL;
837         }

839         expr->ctype = ssize_t_ctype;
840         if (lbase->bit_size > bits_in_char) {
841                 struct expression *sub = alloc_expression(expr->pos, EXPR_BINOP)
842                 struct expression *div = expr;
843                 struct expression *val = alloc_expression(expr->pos, EXPR_VALUE)
844                 unsigned long value = bits_to_bytes(lbase->bit_size);

846                 val->ctype = size_t_ctype;
847                 val->value = value;

849                 if (value & (value-1)) {
850                         if (Wptr_subtraction_blows)
851                                 warning(expr->pos, "potentially expensive pointe
852                 }
```



```
854                 sub->op = '-';
855                 sub->ctype = ssize_t_ctype;
856                 sub->left = l;
857                 sub->right = r;

859                 div->op = '/';
860                 div->left = sub;
861                 div->right = val;
862         }
863
864         return ssize_t_ctype;
865 }

867 #define is_safe_type(type) ((type)->ctype.modifiers & MOD_SAFE)

869 static struct symbol *evaluate_conditional(struct expression *expr, int iterator
870 {
871         struct symbol *ctype;

873         if (!expr)
874                 return NULL;

876         if (!iterator && expr->type == EXPR_ASSIGNMENT && expr->op == '=')
877                 warning(expr->pos, "assignment expression in conditional");

879         ctype = evaluate_expression(expr);
880         if (ctype) {
881                 if (is_safe_type(ctype))
882                         warning(expr->pos, "testing a 'safe expression'");
883                 if (is_func_type(ctype)) {
884                         if (Waddress)
885                                 warning(expr->pos, "the address of %s will alway
886                 } else if (is_array_type(ctype)) {
887                         if (Waddress)
888                                 warning(expr->pos, "the address of %s will alway
889                 } else if (!is_scalar_type(ctype)) {
890                         sparse_error(expr->pos, "incorrect type in conditional")
891                         info(expr->pos, "   got %s", show_typename(ctype));
892                         ctype = NULL;
893                 }
894         }
895         ctype = degenerate(expr);

897         return ctype;
898 }

900 static struct symbol *evaluate_logical(struct expression *expr)
901 {
902         if (!evaluate_conditional(expr->left, 0))
903                 return NULL;
904         if (!evaluate_conditional(expr->right, 0))
905                 return NULL;

907         /* the result is int [6.5.13(3), 6.5.14(3)] */
908         expr->ctype = &int_ctype;
909         expr->flags = expr->left->flags & expr->right->flags;
910         expr->flags &= ~(CEF_CONST_MASK | CEF_ADDR);
911         return &int_ctype;
912 }

914 static struct symbol *evaluate_binop(struct expression *expr)
915 {
916         struct symbol *ltype, *rtype, *ctype;
917         int lclass = classify_type(expr->left->ctype, &ltype);
918         int rclass = classify_type(expr->right->ctype, &rtype);
```

```
919         int op = expr->op;
920
921         /* number op number */
922         if (lclass & rclass & TYPE_NUM) {
923                 expr->flags = expr->left->flags & expr->right->flags;
924                 expr->flags &= ~CEF_CONST_MASK;
925
926                 if ((lclass | rclass) & TYPE_FLOAT) {
927                         switch (op) {
928                         case '+': case '-': case '*': case '/':
929                                 break;
930                         default:
931                                 return bad_expr_type(expr);
932                         }
933                 }
934
935                 if (op == SPECIAL_LEFTSHIFT || op == SPECIAL_RIGHTSHIFT) {
936                         // shifts do integer promotions, but that's it.
937                         unrestrict(expr->left, lclass, &ltype);
938                         unrestrict(expr->right, rclass, &rtype);
939                         ctype = ltype = integer_promotion(ltype);
940                         rtype = integer_promotion(rtype);
941                 } else {
942                         // The rest do usual conversions
943                         const unsigned left_not  = expr->left->type == EXPR_PREO
944                                                    && expr->left->op == '!';
945                         const unsigned right_not = expr->right->type == EXPR_PRE
946                                                    && expr->right->op == '!';
947                         if ((op == '&' || op == '|') && (left_not || right_not))
948                                 warning(expr->pos, "dubious: %sx %c %sy",
949                                         left_not ? "!" : "",
950                                         op,
951                                         right_not ? "!" : "");
952
953                         ltype = usual_conversions(op, expr->left, expr->right,
954                                                   lclass, rclass, ltype, rtype);
955                         ctype = rtype = ltype;
956                 }
957
958                 expr->left = cast_to(expr->left, ltype);
959                 expr->right = cast_to(expr->right, rtype);
960                 expr->ctype = ctype;
961                 return ctype;
962         }
963
964         /* pointer (+|-) integer */
965         if (lclass & TYPE_PTR && is_int(rclass) && (op == '+' || op == '-')) {
966                 unrestrict(expr->right, rclass, &rtype);
967                 return evaluate_ptr_add(expr, rtype);
968         }
969
970         /* integer + pointer */
971         if (rclass & TYPE_PTR && is_int(lclass) && op == '+') {
972                 struct expression *index = expr->left;
973                 unrestrict(index, lclass, &ltype);
974                 expr->left = expr->right;
975                 expr->right = index;
976                 return evaluate_ptr_add(expr, ltype);
977         }
978
979         /* pointer - pointer */
980         if (lclass & rclass & TYPE_PTR && expr->op == '-')
981                 return evaluate_ptr_sub(expr);
982
983         return bad_expr_type(expr);
984 }
```



```
986 static struct symbol *evaluate_comma(struct expression *expr)
987 {
988         expr->ctype = degenerate(expr->right);
989         if (expr->ctype == &null_ctype)
990                 expr->ctype = &ptr_ctype;
991         expr->flags &= expr->left->flags & expr->right->flags;
992         return expr->ctype;
993 }
994
995 static int modify_for_unsigned(int op)
996 {
997         if (op == '<')
998                 op = SPECIAL_UNSIGNED_LT;
999         else if (op == '>')
1000                op = SPECIAL_UNSIGNED_GT;
1001        else if (op == SPECIAL_LTE)
1002                op = SPECIAL_UNSIGNED_LTE;
1003        else if (op == SPECIAL_GTE)
1004                op = SPECIAL_UNSIGNED_GTE;
1005        return op;
1006 }
1007
1008 static inline int is_null_pointer_constant(struct expression *e)
1009 {
1010         if (e->ctype == &null_ctype)
1011                 return 1;
1012         if (!(e->flags & CEF_ICE))
1013                 return 0;
1014         return is_zero_constant(e) ? 2 : 0;
1015 }
1016
1017 static struct symbol *evaluate_compare(struct expression *expr)
1018 {
1019         struct expression *left = expr->left, *right = expr->right;
1020         struct symbol *ltype, *rtype, *lbase, *rbase;
1021         int lclass = classify_type(degenerate(left), &ltype);
1022         int rclass = classify_type(degenerate(right), &rtype);
1023         struct symbol *ctype;
1024         const char *typediff;
1025
1026         /* Type types? */
1027         if (is_type_type(ltype) && is_type_type(rtype)) {
1028                 /*
1029                  * __builtin_types_compatible_p() yields an integer
1030                  * constant expression
1031                  */
1032                 expr->flags = CEF_SET_ICE;
1033                 goto OK;
1034         }
1035
1036         if (is_safe_type(left->ctype) || is_safe_type(right->ctype))
1037                 warning(expr->pos, "testing a 'safe expression'");
1038
1039         expr->flags = left->flags & right->flags & ~CEF_CONST_MASK & ~CEF_ADDR;
1040
1041         /* number on number */
1042         if (lclass & rclass & TYPE_NUM) {
1043                 ctype = usual_conversions(expr->op, expr->left, expr->right,
1044                                           lclass, rclass, ltype, rtype);
1045                 expr->left = cast_to(expr->left, ctype);
1046                 expr->right = cast_to(expr->right, ctype);
1047                 if (ctype->ctype.modifiers & MOD_UNSIGNED)
1048                         expr->op = modify_for_unsigned(expr->op);
1049                 goto OK;
1050         }
```

```
1052         /* at least one must be a pointer */
1053         if (!((lclass | rclass) & TYPE_PTR))
1054                 return bad_expr_type(expr);
1055
1056         /* equality comparisons can be with null pointer constants */
1057         if (expr->op == SPECIAL_EQUAL || expr->op == SPECIAL_NOTEQUAL) {
1058                 int is_null1 = is_null_pointer_constant(left);
1059                 int is_null2 = is_null_pointer_constant(right);
1060                 if (is_null1 == 2)
1061                         bad_null(left);
1062                 if (is_null2 == 2)
1063                         bad_null(right);
1064                 if (is_null1 && is_null2) {
1065                         int positive = expr->op == SPECIAL_EQUAL;
1066                         expr->type = EXPR_VALUE;
1067                         expr->value = positive;
1068                         goto OK;
1069                 }
1070                 if (is_null1 && (rclass & TYPE_PTR)) {
1071                         left = cast_to(left, rtype);
1072                         goto OK;
1073                 }
1074                 if (is_null2 && (lclass & TYPE_PTR)) {
1075                         right = cast_to(right, ltype);
1076                         goto OK;
1077                 }
1078         }
1079         /* both should be pointers */
1080         if (!(lclass & rclass & TYPE_PTR))
1081                 return bad_expr_type(expr);
1082         expr->op = modify_for_unsigned(expr->op);
1083
1084         lbase = examine_pointer_target(ltype);
1085         rbase = examine_pointer_target(rtype);
1086
1087         /* they also have special treatment for pointers to void */
1088         if (expr->op == SPECIAL_EQUAL || expr->op == SPECIAL_NOTEQUAL) {
1089                 if (ltype->ctype.as == rtype->ctype.as) {
1090                         if (lbase == &void_ctype) {
1091                                 right = cast_to(right, ltype);
1092                                 goto OK;
1093                         }
1094                         if (rbase == &void_ctype) {
1095                                 left = cast_to(left, rtype);
1096                                 goto OK;
1097                         }
1098                 }
1099         }
1100
1101         typediff = type_difference(&ltype->ctype, &rtype->ctype,
1102                                    target_qualifiers(rtype),
1103                                    target_qualifiers(ltype));
1104         if (!typediff)
1105                 goto OK;
1106
1107         expression_error(expr, "incompatible types in comparison expression (%s)
1108         return NULL;
1109
1110 OK:
1111         /* the result is int [6.5.8(6), 6.5.9(3)]*/
1112         expr->ctype = &int_ctype;
1113         return &int_ctype;
1114 }
1115
1116 /*
```



```
1117  * NOTE! The degenerate case of "x ? : y", where we don't
1118  * have a true case, this will possibly promote "x" to the
1119  * same type as "y", and thus _change_ the conditional
1120  * test in the expression. But since promotion is "safe"
1121  * for testing, that's OK.
1122  */
1123 static struct symbol *evaluate_conditional_expression(struct expression *expr)
1124 {
1125         struct expression **true;
1126         struct symbol *ctype, *ltype, *rtype, *lbase, *rbase;
1127         int lclass, rclass;
1128         const char * typediff;
1129         int qual;
1130
1131         if (!evaluate_conditional(expr->conditional, 0))
1132                 return NULL;
1133         if (!evaluate_expression(expr->cond_false))
1134                 return NULL;
1135
1136         ctype = degenerate(expr->conditional);
1137         rtype = degenerate(expr->cond_false);
1138
1139         true = &expr->conditional;
1140         ltype = ctype;
1141         if (expr->cond_true) {
1142                 if (!evaluate_expression(expr->cond_true))
1143                         return NULL;
1144                 ltype = degenerate(expr->cond_true);
1145                 true = &expr->cond_true;
1146         }
1147
1148         expr->flags = (expr->conditional->flags & (*true)->flags &
1149                        expr->cond_false->flags & ~CEF_CONST_MASK);
1150         /*
1151          * A conditional operator yields a particular constant
1152          * expression type only if all of its three subexpressions are
1153          * of that type [6.6(6), 6.6(8)].
1154          * As an extension, relax this restriction by allowing any
1155          * constant expression type for the condition expression.
1156          *
1157          * A conditional operator never yields an address constant
1158          * [6.6(9)].
1159          * However, as an extension, if the condition is any constant
1160          * expression, and the true and false expressions are both
1161          * address constants, mark the result as an address constant.
1162          */
1163         if (expr->conditional->flags & (CEF_ACE | CEF_ADDR))
1164                 expr->flags = (*true)->flags & expr->cond_false->flags & ~CEF_CO
1165
1166         lclass = classify_type(ltype, &ltype);
1167         rclass = classify_type(rtype, &rtype);
1168         if (lclass & rclass & TYPE_NUM) {
1169                 ctype = usual_conversions('?', *true, expr->cond_false,
1170                                           lclass, rclass, ltype, rtype);
1171                 *true = cast_to(*true, ctype);
1172                 expr->cond_false = cast_to(expr->cond_false, ctype);
1173                 goto out;
1174         }
1175
1176         if ((lclass | rclass) & TYPE_PTR) {
1177                 int is_null1 = is_null_pointer_constant(*true);
1178                 int is_null2 = is_null_pointer_constant(expr->cond_false);
1179
1180                 if (is_null1 && is_null2) {
1181                         *true = cast_to(*true, &ptr_ctype);
1182                         expr->cond_false = cast_to(expr->cond_false, &ptr_ctype)
```

```
1183                        ctype = &ptr_ctype;
1184                        goto out;
1185                }
1186                if (is_null1 && (rclass & TYPE_PTR)) {
1187                        if (is_null1 == 2)
1188                                bad_null(*true);
1189                        *true = cast_to(*true, rtype);
1190                        ctype = rtype;
1191                        goto out;
1192                }
1193                if (is_null2 && (lclass & TYPE_PTR)) {
1194                        if (is_null2 == 2)
1195                                bad_null(expr->cond_false);
1196                        expr->cond_false = cast_to(expr->cond_false, ltype);
1197                        ctype = ltype;
1198                        goto out;
1199                }
1200                if (!(lclass & rclass & TYPE_PTR)) {
1201                        typediff = "different types";
1202                        goto Err;
1203                }
1204                /* OK, it's pointer on pointer */
1205                if (ltype->ctype.as != rtype->ctype.as) {
1206                        typediff = "different address spaces";
1207                        goto Err;
1208                }

1209                /* need to be lazier here */
1210                lbase = examine_pointer_target(ltype);
1211                rbase = examine_pointer_target(rtype);
1212                qual = target_qualifiers(ltype) | target_qualifiers(rtype);

1213                if (lbase == &void_ctype) {
1214                        /* XXX: pointers to function should warn here */
1215                        ctype = ltype;
1216                        goto Qual;

1217                }
1218                if (rbase == &void_ctype) {
1219                        /* XXX: pointers to function should warn here */
1220                        ctype = rtype;
1221                        goto Qual;
1222                }
1223                /* XXX: that should be pointer to composite */
1224                ctype = ltype;
1225                typediff = type_difference(&ltype->ctype, &rtype->ctype,
1226                                           qual, qual);
1227                if (!typediff)
1228                        goto Qual;
1229                goto Err;
1230        }

1231        /* void on void, struct on same struct, union on same union */
1232        if (ltype == rtype) {
1233                ctype = ltype;
1234                goto out;
1235        }
1236        typediff = "different base types";

1237 Err:
1238        expression_error(expr, "incompatible types in conditional expression (%s
1239        /*
1240         * if the condition is constant, the type is in fact known
1241         * so use it, as gcc & clang do.
1242         */
1243        switch (expr_truth_value(expr->conditional)) {
```



```
1244        case 1: expr->ctype = ltype;
1245                break;
1246        case 0: expr->ctype = rtype;
1247                break;
1248        default:
1249                break;
1250        }
1251        return NULL;

1252 out:
1253        expr->ctype = ctype;
1254        return ctype;

1255 Qual:
1256        if (qual & ~ctype->ctype.modifiers) {
1257                struct symbol *sym = alloc_symbol(ctype->pos, SYM_PTR);
1258                *sym = *ctype;
1259                sym->ctype.modifiers |= qual;
1260                ctype = sym;
1261        }
1262        *true = cast_to(*true, ctype);
1263        expr->cond_false = cast_to(expr->cond_false, ctype);
1264        goto out;
1265 }

1266 /* FP assignments can not do modulo or bit operations */
1267 static int compatible_float_op(int op)
1268 {
1269        return  op == SPECIAL_ADD_ASSIGN ||
1270                op == SPECIAL_SUB_ASSIGN ||
1271                op == SPECIAL_MUL_ASSIGN ||
1272                op == SPECIAL_DIV_ASSIGN;
1273 }

1274 static int evaluate_assign_op(struct expression *expr)
1275 {
1276        struct symbol *target = expr->left->ctype;
1277        struct symbol *source = expr->right->ctype;
1278        struct symbol *t, *s;
1279        int tclass = classify_type(target, &t);
1280        int sclass = classify_type(source, &s);
1281        int op = expr->op;

1282        if (tclass & sclass & TYPE_NUM) {
1283                if (tclass & TYPE_FLOAT && !compatible_float_op(op)) {
1284                        expression_error(expr, "invalid assignment");
1285                        return 0;
1286                }
1287                if (tclass & TYPE_RESTRICT) {
1288                        if (!restricted_binop(op, t)) {
1289                                warning(expr->pos, "bad assignment (%s) to %s",
1290                                        show_special(op), show_typename(t));
1291                                expr->right = cast_to(expr->right, target);
1292                                return 0;
1293                        }
1294                        /* allowed assignments unfoul */
1295                        if (sclass & TYPE_FOULED && unfoul(s) == t)
1296                                goto Cast;
1297                        if (!restricted_value(expr->right, t))
1298                                return 1;
1299                } else if (!(sclass & TYPE_RESTRICT))
1300                        goto usual;
1301                /* source and target would better be identical restricted */
1302                if (t == s)
1303                        return 1;
1304                warning(expr->pos, "invalid assignment: %s", show_special(op));
```

```
1315                 info(expr->pos, "   left side has type %s", show_typename(t));
1316                 info(expr->pos, "   right side has type %s", show_typename(s));
1317                 expr->right = cast_to(expr->right, target);
1318                 return 0;
1319         }
1320         if (tclass == TYPE_PTR && is_int(sclass)) {
1321                 if (op == SPECIAL_ADD_ASSIGN || op == SPECIAL_SUB_ASSIGN) {
1322                         unrestrict(expr->right, sclass, &s);
1323                         evaluate_ptr_add(expr, s);
1324                         return 1;
1325                 }
1326                 expression_error(expr, "invalid pointer assignment");
1327                 return 0;
1328         }

1330         expression_error(expr, "invalid assignment");
1331         return 0;

1333 usual:
1334         target = usual_conversions(op, expr->left, expr->right,
1335                                    tclass, sclass, target, source);
1336 Cast:
1337         expr->right = cast_to(expr->right, target);
1338         return 1;
1339 }

1341 static int whitelist_pointers(struct symbol *t1, struct symbol *t2)
1342 {
1343         if (t1 == t2)
1344                 return 0;       /* yes, 0 - we don't want a cast_to here */
1345         if (t1 == &void_ctype)
1346                 return 1;
1347         if (t2 == &void_ctype)
1348                 return 1;
1349         if (classify_type(t1, &t1) != TYPE_NUM)
1350                 return 0;
1351         if (classify_type(t2, &t2) != TYPE_NUM)
1352                 return 0;
1353         if (t1 == t2)
1354                 return 1;
1355         if (t1->ctype.modifiers & t2->ctype.modifiers & MOD_CHAR)
1356                 return 1;
1357         if ((t1->ctype.modifiers ^ t2->ctype.modifiers) & MOD_SIZE)
1358                 return 0;
1359         return !Wtypesign;
1360 }

1362 static int check_assignment_types(struct symbol *target, struct expression **rp,
1363         const char **typediff)
1364 {
1365         struct symbol *source = degenerate(*rp);
1366         struct symbol *t, *s;
1367         int tclass = classify_type(target, &t);
1368         int sclass = classify_type(source, &s);

1370         if (tclass & sclass & TYPE_NUM) {
1371                 if (tclass & TYPE_RESTRICT) {
1372                         /* allowed assignments unfoul */
1373                         if (sclass & TYPE_FOULED && unfoul(s) == t)
1374                                 goto Cast;
1375                         if (!restricted_value(*rp, target))
1376                                 return 1;
1377                         if (s == t)
1378                                 return 1;
1379                 } else if (!(sclass & TYPE_RESTRICT))
1380                         goto Cast;
```



```
1381                 if (t == &bool_ctype) {
1382                         if (is_fouled_type(s))
1383                                 warning((*rp)->pos, "%s degrades to integer",
1384                                         show_typename(s->ctype.base_type));
1385                         goto Cast;
1386                 }
1387                 *typediff = "different base types";
1388                 return 0;
1389         }

1391         if (tclass == TYPE_PTR) {
1392                 unsigned long mod1, mod2;
1393                 struct symbol *b1, *b2;
1394                 // NULL pointer is always OK
1395                 int is_null = is_null_pointer_constant(*rp);
1396                 if (is_null) {
1397                         if (is_null == 2)
1398                                 bad_null(*rp);
1399                         goto Cast;
1400                 }
1401                 if (!(sclass & TYPE_PTR)) {
1402                         *typediff = "different base types";
1403                         return 0;
1404                 }
1405                 b1 = examine_pointer_target(t);
1406                 b2 = examine_pointer_target(s);
1407                 mod1 = target_qualifiers(t);
1408                 mod2 = target_qualifiers(s);
1409                 if (whitelist_pointers(b1, b2)) {
1410                         /*
1411                          * assignments to/from void * are OK, provided that
1412                          * we do not remove qualifiers from pointed to [C]
1413                          * or mix address spaces [sparse].
1414                          */
1415                         if (t->ctype.as != s->ctype.as) {
1416                                 *typediff = "different address spaces";
1417                                 return 0;
1418                         }
1419                         /*
1420                          * If this is a function pointer assignment, it is
1421                          * actually fine to assign a pointer to const data to
1422                          * it, as a function pointer points to const data
1423                          * implicitly, i.e., dereferencing it does not produce
1424                          * an lvalue.
1425                          */
1426                         if (b1->type == SYM_FN)
1427                                 mod1 |= MOD_CONST;
1428                         if (mod2 & ~mod1) {
1429                                 *typediff = "different modifiers";
1430                                 return 0;
1431                         }
1432                         goto Cast;
1433                 }
1434                 /* It's OK if the target is more volatile or const than the sour
1435                 *typediff = type_difference(&t->ctype, &s->ctype, 0, mod1);
1436                 if (*typediff)
1437                         return 0;
1438                 return 1;
1439         }

1441         if ((tclass & TYPE_COMPOUND) && s == t)
1442                 return 1;

1444         if (tclass & TYPE_NUM) {
1445                 /* XXX: need to turn into comparison with NULL */
1446                 if (t == &bool_ctype && (sclass & TYPE_PTR))
```

```
1447                        goto Cast;
1448                *typediff = "different base types";
1449                return 0;
1450        }
1451        *typediff = "invalid types";
1452        return 0;

1454 Cast:
1455        *rp = cast_to(*rp, target);
1456        return 1;
1457 }

1459 static int compatible_assignment_types(struct expression *expr, struct symbol *t
1460        struct expression **rp, const char *where)
1461 {
1462        const char *typediff;
1463        struct symbol *source = degenerate(*rp);

1465        if (!check_assignment_types(target, rp, &typediff)) {
1466                warning(expr->pos, "incorrect type in %s (%s)", where, typediff)
1467                info(expr->pos, "   expected %s", show_typename(target));
1468                info(expr->pos, "   got %s", show_typename(source));
1469                *rp = cast_to(*rp, target);
1470                return 0;
1471        }

1473        return 1;
1474 }

1476 static int compatible_transparent_union(struct symbol *target,
1477        struct expression **rp)
1478 {
1479        struct symbol *t, *member;
1480        classify_type(target, &t);
1481        if (t->type != SYM_UNION || !t->transparent_union)
1482                return 0;

1484        FOR_EACH_PTR(t->symbol_list, member) {
1485                const char *typediff;
1486                if (check_assignment_types(member, rp, &typediff))
1487                        return 1;
1488        } END_FOR_EACH_PTR(member);

1490        return 0;
1491 }

1493 static int compatible_argument_type(struct expression *expr, struct symbol *targ
1494        struct expression **rp, const char *where)
1495 {
1496        if (compatible_transparent_union(target, rp))
1497                return 1;

1499        return compatible_assignment_types(expr, target, rp, where);
1500 }

1502 static void mark_assigned(struct expression *expr)
1503 {
1504        struct symbol *sym;

1506        if (!expr)
1507                return;
1508        switch (expr->type) {
1509        case EXPR_SYMBOL:
1510                sym = expr->symbol;
1511                if (!sym)
1512                        return;
```



```
1513                if (sym->type != SYM_NODE)
1514                        return;
1515                sym->ctype.modifiers |= MOD_ASSIGNED;
1516                return;

1518        case EXPR_BINOP:
1519                mark_assigned(expr->left);
1520                mark_assigned(expr->right);
1521                return;
1522        case EXPR_CAST:
1523        case EXPR_FORCE_CAST:
1524                mark_assigned(expr->cast_expression);
1525                return;
1526        case EXPR_SLICE:
1527                mark_assigned(expr->base);
1528                return;
1529        default:
1530                /* Hmm? */
1531                return;
1532        }
1533 }

1535 static void evaluate_assign_to(struct expression *left, struct symbol *type)
1536 {
1537        if (type->ctype.modifiers & MOD_CONST)
1538                expression_error(left, "assignment to const expression");

1540        /* We know left is an lvalue, so it's a "preop-*" */
1541        mark_assigned(left->unop);
1542 }

1544 static struct symbol *evaluate_assignment(struct expression *expr)
1545 {
1546        struct expression *left = expr->left;
1547        struct expression *where = expr;
1548        struct symbol *ltype;

1550        if (!lvalue_expression(left)) {
1551                expression_error(expr, "not an lvalue");
1552                return NULL;
1553        }

1555        ltype = left->ctype;

1557        if (expr->op != '=') {
1558                if (!evaluate_assign_op(expr))
1559                        return NULL;
1560        } else {
1561                if (!compatible_assignment_types(where, ltype, &expr->right, "as
1562                        return NULL;
1563        }

1565        evaluate_assign_to(left, ltype);

1567        expr->ctype = ltype;
1568        return ltype;
1569 }

1571 static void examine_fn_arguments(struct symbol *fn)
1572 {
1573        struct symbol *s;

1575        FOR_EACH_PTR(fn->arguments, s) {
1576                struct symbol *arg = evaluate_symbol(s);
1577                /* Array/function arguments silently degenerate into pointers */
1578                if (arg) {
```

```
1579                        struct symbol *ptr;
1580                        switch(arg->type) {
1581                        case SYM_ARRAY:
1582                        case SYM_FN:
1583                                ptr = alloc_symbol(s->pos, SYM_PTR);
1584                                if (arg->type == SYM_ARRAY)
1585                                        ptr->ctype = arg->ctype;
1586                                else
1587                                        ptr->ctype.base_type = arg;
1588                                ptr->ctype.as |= s->ctype.as;
1589                                ptr->ctype.modifiers |= s->ctype.modifiers & MOD

1591                                s->ctype.base_type = ptr;
1592                                s->ctype.as = 0;
1593                                s->ctype.modifiers &= ~MOD_PTRINHERIT;
1594                                s->bit_size = 0;
1595                                s->examined = 0;
1596                                examine_symbol_type(s);
1597                                break;
1598                        default:
1599                                /* nothing */
1600                                break;
1601                        }
1602                }
1603        } END_FOR_EACH_PTR(s);
1604 }

1606 static struct symbol *convert_to_as_mod(struct symbol *sym, int as, int mod)
1607 {
1608         /* Take the modifiers of the pointer, and apply them to the member */
1609         mod |= sym->ctype.modifiers;
1610         if (sym->ctype.as != as || sym->ctype.modifiers != mod) {
1611                 struct symbol *newsym = alloc_symbol(sym->pos, SYM_NODE);
1612                 *newsym = *sym;
1613                 newsym->ctype.as = as;
1614                 newsym->ctype.modifiers = mod;
1615                 sym = newsym;
1616         }
1617         return sym;
1618 }

1620 static struct symbol *create_pointer(struct expression *expr, struct symbol *sym
1621 {
1622         struct symbol *node = alloc_symbol(expr->pos, SYM_NODE);
1623         struct symbol *ptr = alloc_symbol(expr->pos, SYM_PTR);

1625         node->ctype.base_type = ptr;
1626         ptr->bit_size = bits_in_pointer;
1627         ptr->ctype.alignment = pointer_alignment;

1629         node->bit_size = bits_in_pointer;
1630         node->ctype.alignment = pointer_alignment;

1632         access_symbol(sym);
1633         if (sym->ctype.modifiers & MOD_REGISTER) {
1634                 warning(expr->pos, "taking address of 'register' variable '%s'",
1635                 sym->ctype.modifiers &= ~MOD_REGISTER;
1636         }
1637         if (sym->type == SYM_NODE) {
1638                 ptr->ctype.as |= sym->ctype.as;
1639                 ptr->ctype.modifiers |= sym->ctype.modifiers & MOD_PTRINHERIT;
1640                 sym = sym->ctype.base_type;
1641         }
1642         if (degenerate && sym->type == SYM_ARRAY) {
1643                 ptr->ctype.as |= sym->ctype.as;
1644                 ptr->ctype.modifiers |= sym->ctype.modifiers & MOD_PTRINHERIT;
```



```
1645                 sym = sym->ctype.base_type;
1646         }
1647         ptr->ctype.base_type = sym;

1649         return node;
1650 }

1652 /* Arrays degenerate into pointers on pointer arithmetic */
1653 static struct symbol *degenerate(struct expression *expr)
1654 {
1655         struct symbol *ctype, *base;

1657         if (!expr)
1658                 return NULL;
1659         ctype = expr->ctype;
1660         if (!ctype)
1661                 return NULL;
1662         base = examine_symbol_type(ctype);
1663         if (ctype->type == SYM_NODE)
1664                 base = ctype->ctype.base_type;
1665         /*
1666          * Arrays degenerate into pointers to the entries, while
1667          * functions degenerate into pointers to themselves.
1668          * If array was part of non-lvalue compound, we create a copy
1669          * of that compound first and then act as if we were dealing with
1670          * the corresponding field in there.
1671          */
1672         switch (base->type) {
1673         case SYM_ARRAY:
1674                 if (expr->type == EXPR_SLICE) {
1675                         struct symbol *a = alloc_symbol(expr->pos, SYM_NODE);
1676                         struct expression *e0, *e1, *e2, *e3, *e4;

1678                         a->ctype.base_type = expr->base->ctype;
1679                         a->bit_size = expr->base->ctype->bit_size;
1680                         a->array_size = expr->base->ctype->array_size;

1682                         e0 = alloc_expression(expr->pos, EXPR_SYMBOL);
1683                         e0->symbol = a;
1684                         e0->ctype = &lazy_ptr_ctype;

1686                         e1 = alloc_expression(expr->pos, EXPR_PREOP);
1687                         e1->unop = e0;
1688                         e1->op = '*';
1689                         e1->ctype = expr->base->ctype;  /* XXX */

1691                         e2 = alloc_expression(expr->pos, EXPR_ASSIGNMENT);
1692                         e2->left = e1;
1693                         e2->right = expr->base;
1694                         e2->op = '=';
1695                         e2->ctype = expr->base->ctype;

1697                         if (expr->r_bitpos) {
1698                                 e3 = alloc_expression(expr->pos, EXPR_BINOP);
1699                                 e3->op = '+';
1700                                 e3->left = e0;
1701                                 e3->right = alloc_const_expression(expr->pos,
1702                                                 bits_to_bytes(expr->r_bi
1703                                 e3->ctype = &lazy_ptr_ctype;
1704                         } else {
1705                                 e3 = e0;
1706                         }

1708                         e4 = alloc_expression(expr->pos, EXPR_COMMA);
1709                         e4->left = e2;
1710                         e4->right = e3;
```

```
1711                        e4->ctype = &lazy_ptr_ctype;

1713                        expr->unop = e4;
1714                        expr->type = EXPR_PREOP;
1715                        expr->op = '*';
1716                }
1717        case SYM_FN:
1718                if (expr->op != '*' || expr->type != EXPR_PREOP) {
1719                        expression_error(expr, "strange non-value function or ar
1720                        return &bad_ctype;
1721                }
1722                *expr = *expr->unop;
1723                ctype = create_pointer(expr, ctype, 1);
1724                expr->ctype = ctype;
1725        default:
1726                /* nothing */;
1727        }
1728        return ctype;
1729 }

1731 static struct symbol *evaluate_addressof(struct expression *expr)
1732 {
1733        struct expression *op = expr->unop;
1734        struct symbol *ctype;

1736        if (op->op != '*' || op->type != EXPR_PREOP) {
1737                expression_error(expr, "not addressable");
1738                return NULL;
1739        }
1740        ctype = op->ctype;
1741        *expr = *op->unop;

1743        if (expr->type == EXPR_SYMBOL) {
1744                struct symbol *sym = expr->symbol;
1745                sym->ctype.modifiers |= MOD_ADDRESSABLE;
1746        }

1748        /*
1749         * symbol expression evaluation is lazy about the type
1750         * of the sub-expression, so we may have to generate
1751         * the type here if so..
1752         */
1753        if (expr->ctype == &lazy_ptr_ctype) {
1754                ctype = create_pointer(expr, ctype, 0);
1755                expr->ctype = ctype;
1756        }
1757        return expr->ctype;
1758 }


1761 static struct symbol *evaluate_dereference(struct expression *expr)
1762 {
1763        struct expression *op = expr->unop;
1764        struct symbol *ctype = op->ctype, *node, *target;

1766        /* Simplify: *&(expr) => (expr) */
1767        if (op->type == EXPR_PREOP && op->op == '&') {
1768                *expr = *op->unop;
1769                expr->flags = CEF_NONE;
1770                return expr->ctype;
1771        }

1773        examine_symbol_type(ctype);

1775        /* Dereferencing a node drops all the node information. */
1776        if (ctype->type == SYM_NODE)
```



```
1777                ctype = ctype->ctype.base_type;

1779        node = alloc_symbol(expr->pos, SYM_NODE);
1780        target = ctype->ctype.base_type;

1782        switch (ctype->type) {
1783        default:
1784                expression_error(expr, "cannot dereference this type");
1785                return NULL;
1786        case SYM_PTR:
1787                node->ctype.modifiers = target->ctype.modifiers & MOD_SPECIFIER;
1788                merge_type(node, ctype);
1789                break;

1791        case SYM_ARRAY:
1792                if (!lvalue_expression(op)) {
1793                        expression_error(op, "non-lvalue array??");
1794                        return NULL;
1795                }

1797                /* Do the implied "addressof" on the array */
1798                *op = *op->unop;

1800                /*
1801                 * When an array is dereferenced, we need to pick
1802                 * up the attributes of the original node too..
1803                 */
1804                merge_type(node, op->ctype);
1805                merge_type(node, ctype);
1806                break;
1807        }

1809        node->bit_size = target->bit_size;
1810        node->array_size = target->array_size;

1812        expr->ctype = node;
1813        return node;
1814 }

1816 /*
1817  * Unary post-ops: x++ and x--
1818  */
1819 static struct symbol *evaluate_postop(struct expression *expr)
1820 {
1821        struct expression *op = expr->unop;
1822        struct symbol *ctype = op->ctype;
1823        int class = classify_type(ctype, &ctype);
1824        int multiply = 0;

1826        if (!class || class & TYPE_COMPOUND) {
1827                expression_error(expr, "need scalar for ++/--");
1828                return NULL;
1829        }
1830        if (!lvalue_expression(expr->unop)) {
1831                expression_error(expr, "need lvalue expression for ++/--");
1832                return NULL;
1833        }

1835        if ((class & TYPE_RESTRICT) && restricted_unop(expr->op, &ctype))
1836                unrestrict(expr, class, &ctype);

1838        if (class & TYPE_NUM) {
1839                multiply = 1;
1840        } else if (class == TYPE_PTR) {
1841                struct symbol *target = examine_pointer_target(ctype);
1842                if (!is_function(target))
```

```
1843                        multiply = bits_to_bytes(target->bit_size);
1844        }

1846        if (multiply) {
1847                evaluate_assign_to(op, op->ctype);
1848                expr->op_value = multiply;
1849                expr->ctype = ctype;
1850                return ctype;
1851        }

1853        expression_error(expr, "bad argument type for ++/--");
1854        return NULL;
1855 }

1857 static struct symbol *evaluate_sign(struct expression *expr)
1858 {
1859        struct symbol *ctype = expr->unop->ctype;
1860        int class = classify_type(ctype, &ctype);
1861        unsigned char flags = expr->unop->flags & ~CEF_CONST_MASK;

1863        /* should be an arithmetic type */
1864        if (!(class & TYPE_NUM))
1865                return bad_expr_type(expr);
1866        if (class & TYPE_RESTRICT)
1867                goto Restr;
1868 Normal:
1869        if (!(class & TYPE_FLOAT)) {
1870                ctype = integer_promotion(ctype);
1871                expr->unop = cast_to(expr->unop, ctype);
1872        } else if (expr->op != '~') {
1873                /* no conversions needed */
1874        } else {
1875                return bad_expr_type(expr);
1876        }
1877        if (expr->op == '+')
1878                *expr = *expr->unop;
1879        expr->flags = flags;
1880        expr->ctype = ctype;
1881        return ctype;
1882 Restr:
1883        if (restricted_unop(expr->op, &ctype))
1884                unrestrict(expr, class, &ctype);
1885        goto Normal;
1886 }

1888 static struct symbol *evaluate_preop(struct expression *expr)
1889 {
1890        struct symbol *ctype = expr->unop->ctype;

1892        switch (expr->op) {
1893        case '(':
1894                *expr = *expr->unop;
1895                return ctype;

1897        case '+':
1898        case '-':
1899        case '~':
1900                return evaluate_sign(expr);

1902        case '*':
1903                return evaluate_dereference(expr);

1905        case '&':
1906                return evaluate_addressof(expr);

1908        case SPECIAL_INCREMENT:
```



```
1909        case SPECIAL_DECREMENT:
1910                /*
1911                 * From a type evaluation standpoint the preops are
1912                 * the same as the postops
1913                 */
1914                return evaluate_postop(expr);

1916        case '!':
1917                expr->flags = expr->unop->flags & ~CEF_CONST_MASK;
1918                /*
1919                 * A logical negation never yields an address constant
1920                 * [6.6(9)].
1921                 */
1922                expr->flags &= ~CEF_ADDR;

1924                if (is_safe_type(ctype))
1925                        warning(expr->pos, "testing a 'safe expression'");
1926                if (is_float_type(ctype)) {
1927                        struct expression *arg = expr->unop;
1928                        expr->type = EXPR_COMPARE;
1929                        expr->op = SPECIAL_EQUAL;
1930                        expr->left = arg;
1931                        expr->right = alloc_expression(expr->pos, EXPR_FVALUE);
1932                        expr->right->ctype = ctype;
1933                        expr->right->fvalue = 0;
1934                } else if (is_fouled_type(ctype)) {
1935                        warning(expr->pos, "%s degrades to integer",
1936                                show_typename(ctype->ctype.base_type));
1937                }
1938                /* the result is int [6.5.3.3(5)]*/
1939                ctype = &int_ctype;
1940                break;

1942        default:
1943                break;
1944        }
1945        expr->ctype = ctype;
1946        return ctype;
1947 }

1949 static struct symbol *find_identifier(struct ident *ident, struct symbol_list *_
1950 {
1951        struct ptr_list *head = (struct ptr_list *)_list;
1952        struct ptr_list *list = head;

1954        if (!head)
1955                return NULL;
1956        do {
1957                int i;
1958                for (i = 0; i < list->nr; i++) {
1959                        struct symbol *sym = (struct symbol *) list->list[i];
1960                        if (sym->ident) {
1961                                if (sym->ident != ident)
1962                                        continue;
1963                                *offset = sym->offset;
1964                                return sym;
1965                        } else {
1966                                struct symbol *ctype = sym->ctype.base_type;
1967                                struct symbol *sub;
1968                                if (!ctype)
1969                                        continue;
1970                                if (ctype->type != SYM_UNION && ctype->type != S
1971                                        continue;
1972                                sub = find_identifier(ident, ctype->symbol_list,
1973                                if (!sub)
1974                                        continue;
```

```
1975                                *offset += sym->offset;
1976                                return sub;
1977                        }
1978                }
1979        } while ((list = list->next) != head);
1980        return NULL;
1981 }

1983 static struct expression *evaluate_offset(struct expression *expr, unsigned long
1984 {
1985        struct expression *add;

1987        /*
1988         * Create a new add-expression
1989         *
1990         * NOTE! Even if we just add zero, we need a new node
1991         * for the member pointer, since it has a different
1992         * type than the original pointer. We could make that
1993         * be just a cast, but the fact is, a node is a node,
1994         * so we might as well just do the "add zero" here.
1995         */
1996        add = alloc_expression(expr->pos, EXPR_BINOP);
1997        add->op = '+';
1998        add->left = expr;
1999        add->right = alloc_expression(expr->pos, EXPR_VALUE);
2000        add->right->ctype = &int_ctype;
2001        add->right->value = offset;

2003        /*
2004         * The ctype of the pointer will be lazily evaluated if
2005         * we ever take the address of this member dereference..
2006         */
2007        add->ctype = &lazy_ptr_ctype;
2008        /*
2009         * The resulting address of a member access through an address
2010         * constant is an address constant again [6.6(9)].
2011         */
2012        add->flags = expr->flags;

2014        return add;
2015 }

2017 /* structure/union dereference */
2018 static struct symbol *evaluate_member_dereference(struct expression *expr)
2019 {
2020        int offset;
2021        struct symbol *ctype, *member;
2022        struct expression *deref = expr->deref, *add;
2023        struct ident *ident = expr->member;
2024        unsigned int mod;
2025        int address_space;

2027        if (!evaluate_expression(deref))
2028                return NULL;
2029        if (!ident) {
2030                expression_error(expr, "bad member name");
2031                return NULL;
2032        }

2034        ctype = deref->ctype;
2035        examine_symbol_type(ctype);
2036        address_space = ctype->ctype.as;
2037        mod = ctype->ctype.modifiers;
2038        if (ctype->type == SYM_NODE) {
2039                ctype = ctype->ctype.base_type;
2040                address_space |= ctype->ctype.as;
```



```
2041                mod |= ctype->ctype.modifiers;
2042        }
2043        if (!ctype || (ctype->type != SYM_STRUCT && ctype->type != SYM_UNION)) {
2044                expression_error(expr, "expected structure or union");
2045                return NULL;
2046        }
2047        offset = 0;
2048        member = find_identifier(ident, ctype->symbol_list, &offset);
2049        if (!member) {
2050                const char *type = ctype->type == SYM_STRUCT ? "struct" : "union
2051                const char *name = "<unnamed>";
2052                int namelen = 9;
2053                if (ctype->ident) {
2054                        name = ctype->ident->name;
2055                        namelen = ctype->ident->len;
2056                }
2057                if (ctype->symbol_list)
2058                        expression_error(expr, "no member '%s' in %s %.*s",
2059                                show_ident(ident), type, namelen, name);
2060                else
2061                        expression_error(expr, "using member '%s' in "
2062                                "incomplete %s %.*s", show_ident(ident),
2063                                type, namelen, name);
2064                return NULL;
2065        }

2067        /*
2068         * The member needs to take on the address space and modifiers of
2069         * the "parent" type.
2070         */
2071        member = convert_to_as_mod(member, address_space, mod);
2072        ctype = get_base_type(member);

2074        if (!lvalue_expression(deref)) {
2075                if (deref->type != EXPR_SLICE) {
2076                        expr->base = deref;
2077                        expr->r_bitpos = 0;
2078                } else {
2079                        expr->base = deref->base;
2080                        expr->r_bitpos = deref->r_bitpos;
2081                }
2082                expr->r_bitpos += bytes_to_bits(offset);
2083                expr->type = EXPR_SLICE;
2084                expr->r_nrbits = member->bit_size;
2085                expr->r_bitpos += member->bit_offset;
2086                expr->ctype = member;
2087                return member;
2088        }

2090        deref = deref->unop;
2091        expr->deref = deref;

2093        add = evaluate_offset(deref, offset);
2094        expr->type = EXPR_PREOP;
2095        expr->op = '*';
2096        expr->unop = add;

2098        expr->ctype = member;
2099        return member;
2100 }

2102 static int is_promoted(struct expression *expr)
2103 {
2104        while (1) {
2105                switch (expr->type) {
2106                case EXPR_BINOP:
```

```
2107                 case EXPR_SELECT:
2108                 case EXPR_CONDITIONAL:
2109                         return 1;
2110                 case EXPR_COMMA:
2111                         expr = expr->right;
2112                         continue;
2113                 case EXPR_PREOP:
2114                         switch (expr->op) {
2115                         case '(':
2116                                 expr = expr->unop;
2117                                 continue;
2118                         case '+':
2119                         case '-':
2120                         case '~':
2121                                 return 1;
2122                         default:
2123                                 return 0;
2124                         }
2125                 default:
2126                         return 0;
2127                 }
2128         }
2129 }


2132 static struct symbol *evaluate_cast(struct expression *);

2134 static struct symbol *evaluate_type_information(struct expression *expr)
2135 {
2136         struct symbol *sym = expr->cast_type;
2137         if (!sym) {
2138                 sym = evaluate_expression(expr->cast_expression);
2139                 if (!sym)
2140                         return NULL;
2141                 /*
2142                  * Expressions of restricted types will possibly get
2143                  * promoted - check that here
2144                  */
2145                 if (is_restricted_type(sym)) {
2146                         if (sym->bit_size < bits_in_int && is_promoted(expr))
2147                                 sym = &int_ctype;
2148                 } else if (is_fouled_type(sym)) {
2149                         sym = &int_ctype;
2150                 }
2151         }
2152         examine_symbol_type(sym);
2153         if (is_bitfield_type(sym)) {
2154                 expression_error(expr, "trying to examine bitfield type");
2155                 return NULL;
2156         }
2157         return sym;
2158 }

2160 static struct symbol *evaluate_sizeof(struct expression *expr)
2161 {
2162         struct symbol *type;
2163         int size;

2165         type = evaluate_type_information(expr);
2166         if (!type)
2167                 return NULL;

2169         size = type->bit_size;

2171         if (size < 0 && is_void_type(type)) {
2172                 if (Wpointer_arith)
```



```
2173                         warning(expr->pos, "expression using sizeof(void)");
2174                 size = bits_in_char;
2175         }

2177         if (size == 1 && is_bool_type(type)) {
2178                 if (Wsizeof_bool)
2179                         warning(expr->pos, "expression using sizeof bool");
2180                 size = bits_in_char;
2181         }

2183         if (is_function(type->ctype.base_type)) {
2184                 if (Wpointer_arith)
2185                         warning(expr->pos, "expression using sizeof on a functio
2186                 size = bits_in_char;
2187         }

2189         if ((size < 0) || (size & (bits_in_char - 1)))
2190                 expression_error(expr, "cannot size expression");

2192         expr->type = EXPR_VALUE;
2193         expr->value = bits_to_bytes(size);
2194         expr->taint = 0;
2195         expr->ctype = size_t_ctype;
2196         return size_t_ctype;
2197 }

2199 static struct symbol *evaluate_ptrsizeof(struct expression *expr)
2200 {
2201         struct symbol *type;
2202         int size;

2204         type = evaluate_type_information(expr);
2205         if (!type)
2206                 return NULL;

2208         if (type->type == SYM_NODE)
2209                 type = type->ctype.base_type;
2210         if (!type)
2211                 return NULL;
2212         switch (type->type) {
2213         case SYM_ARRAY:
2214                 break;
2215         case SYM_PTR:
2216                 type = get_base_type(type);
2217                 if (type)
2218                         break;
2219         default:
2220                 expression_error(expr, "expected pointer expression");
2221                 return NULL;
2222         }
2223         size = type->bit_size;
2224         if (size & (bits_in_char-1))
2225                 size = 0;
2226         expr->type = EXPR_VALUE;
2227         expr->value = bits_to_bytes(size);
2228         expr->taint = 0;
2229         expr->ctype = size_t_ctype;
2230         return size_t_ctype;
2231 }

2233 static struct symbol *evaluate_alignof(struct expression *expr)
2234 {
2235         struct symbol *type;

2237         type = evaluate_type_information(expr);
2238         if (!type)
```

```
2239                 return NULL;

2241         expr->type = EXPR_VALUE;
2242         expr->value = type->ctype.alignment;
2243         expr->taint = 0;
2244         expr->ctype = size_t_ctype;
2245         return size_t_ctype;
2246 }

2248 static int evaluate_arguments(struct symbol *fn, struct expression_list *head)
2249 {
2250         struct expression *expr;
2251         struct symbol_list *argument_types = fn->arguments;
2252         struct symbol *argtype;
2253         int i = 1;

2255         PREPARE_PTR_LIST(argument_types, argtype);
2256         FOR_EACH_PTR (head, expr) {
2257                 struct expression **p = THIS_ADDRESS(expr);
2258                 struct symbol *ctype, *target;
2259                 ctype = evaluate_expression(expr);

2261                 if (!ctype)
2262                         return 0;

2264                 target = argtype;
2265                 if (!target) {
2266                         struct symbol *type;
2267                         int class = classify_type(ctype, &type);
2268                         if (is_int(class)) {
2269                                 *p = cast_to(expr, integer_promotion(type));
2270                         } else if (class & TYPE_FLOAT) {
2271                                 unsigned long mod = type->ctype.modifiers;
2272                                 if (!(mod & (MOD_LONG_ALL)))
2273                                         *p = cast_to(expr, &double_ctype);
2274                         } else if (class & TYPE_PTR) {
2275                                 if (expr->ctype == &null_ctype)
2276                                         *p = cast_to(expr, &ptr_ctype);
2277                                 else
2278                                         degenerate(expr);
2279                         }
2280                 } else if (!target->forced_arg){
2281                         static char where[30];
2282                         examine_symbol_type(target);
2283                         sprintf(where, "argument %d", i);
2284                         compatible_argument_type(expr, target, p, where);
2285                 }

2287                 i++;
2288                 NEXT_PTR_LIST(argtype);
2289         } END_FOR_EACH_PTR(expr);
2290         FINISH_PTR_LIST(argtype);
2291         return 1;
2292 }

2294 static void convert_index(struct expression *e)
2295 {
2296         struct expression *child = e->idx_expression;
2297         unsigned from = e->idx_from;
2298         unsigned to = e->idx_to + 1;
2299         e->type = EXPR_POS;
2300         e->init_offset = from * bits_to_bytes(e->ctype->bit_size);
2301         e->init_nr = to - from;
2302         e->init_expr = child;
2303 }
```



```
2305 static void convert_ident(struct expression *e)
2306 {
2307         struct expression *child = e->ident_expression;
2308         int offset = e->offset;

2310         e->type = EXPR_POS;
2311         e->init_offset = offset;
2312         e->init_nr = 1;
2313         e->init_expr = child;
2314 }

2316 static void convert_designators(struct expression *e)
2317 {
2318         while (e) {
2319                 if (e->type == EXPR_INDEX)
2320                         convert_index(e);
2321                 else if (e->type == EXPR_IDENTIFIER)
2322                         convert_ident(e);
2323                 else
2324                         break;
2325                 e = e->init_expr;
2326         }
2327 }

2329 static void excess(struct expression *e, const char *s)
2330 {
2331         warning(e->pos, "excessive elements in %s initializer", s);
2332 }

2334 /*
2335  * implicit designator for the first element
2336  */
2337 static struct expression *first_subobject(struct symbol *ctype, int class,
2338                                           struct expression **v)
2339 {
2340         struct expression *e = *v, *new;

2342         if (ctype->type == SYM_NODE)
2343                 ctype = ctype->ctype.base_type;

2345         if (class & TYPE_PTR) { /* array */
2346                 if (!ctype->bit_size)
2347                         return NULL;
2348                 new = alloc_expression(e->pos, EXPR_INDEX);
2349                 new->idx_expression = e;
2350                 new->ctype = ctype->ctype.base_type;
2351         } else  {
2352                 struct symbol *field, *p;
2353                 PREPARE_PTR_LIST(ctype->symbol_list, p);
2354                 while (p && !p->ident && is_bitfield_type(p))
2355                         NEXT_PTR_LIST(p);
2356                 field = p;
2357                 FINISH_PTR_LIST(p);
2358                 if (!field)
2359                         return NULL;
2360                 new = alloc_expression(e->pos, EXPR_IDENTIFIER);
2361                 new->ident_expression = e;
2362                 new->field = new->ctype = field;
2363                 new->offset = field->offset;
2364         }
2365         *v = new;
2366         return new;
2367 }

2369 /*
2370  * sanity-check explicit designators; return the innermost one or NULL
```

```
 * in case of error.  Assign types.
 */
static struct expression *check_designators(struct expression *e,
                                            struct symbol *ctype)
{
        struct expression *last = NULL;
        const char *err;
        while (1) {
                if (ctype->type == SYM_NODE)
                        ctype = ctype->ctype.base_type;
                if (e->type == EXPR_INDEX) {
                        struct symbol *type;
                        if (ctype->type != SYM_ARRAY) {
                                err = "array index in non-array";
                                break;
                        }
                        type = ctype->ctype.base_type;
                        if (ctype->bit_size >= 0 && type->bit_size >= 0) {
                                unsigned offset = array_element_offset(type->bit
                                if (offset >= ctype->bit_size) {
                                        err = "index out of bounds in";
                                        break;
                                }
                        }
                        e->ctype = ctype = type;
                        ctype = type;
                        last = e;
                        if (!e->idx_expression) {
                                err = "invalid";
                                break;
                        }
                        e = e->idx_expression;
                } else if (e->type == EXPR_IDENTIFIER) {
                        int offset = 0;
                        if (ctype->type != SYM_STRUCT && ctype->type != SYM_UNIO
                                err = "field name not in struct or union";
                                break;
                        }
                        ctype = find_identifier(e->expr_ident, ctype->symbol_lis
                        if (!ctype) {
                                err = "unknown field name in";
                                break;
                        }
                        e->offset = offset;
                        e->field = e->ctype = ctype;
                        last = e;
                        if (!e->ident_expression) {
                                err = "invalid";
                                break;
                        }
                        e = e->ident_expression;
                } else if (e->type == EXPR_POS) {
                        err = "internal front-end error: EXPR_POS in";
                        break;
                } else
                        return last;
        }
        expression_error(e, "%s initializer", err);
        return NULL;
}

/*
 * choose the next subobject to initialize.
 *
 * Get designators for next element, switch old ones to EXPR_POS.
 * Return the resulting expression or NULL if we'd run out of subobjects.
```



```
 * The innermost designator is returned in *v.  Designators in old
 * are assumed to be already sanity-checked.
 */
static struct expression *next_designators(struct expression *old,
                                struct symbol *ctype,
                                struct expression *e, struct expression **v)
{
        struct expression *new = NULL;

        if (!old)
                return NULL;
        if (old->type == EXPR_INDEX) {
                struct expression *copy;
                unsigned n;

                copy = next_designators(old->idx_expression,
                                        old->ctype, e, v);
                if (!copy) {
                        n = old->idx_to + 1;
                        if (array_element_offset(old->ctype->bit_size, n) == cty
                                convert_index(old);
                                return NULL;
                        }
                        copy = e;
                        *v = new = alloc_expression(e->pos, EXPR_INDEX);
                } else {
                        n = old->idx_to;
                        new = alloc_expression(e->pos, EXPR_INDEX);
                }

                new->idx_from = new->idx_to = n;
                new->idx_expression = copy;
                new->ctype = old->ctype;
                convert_index(old);
        } else if (old->type == EXPR_IDENTIFIER) {
                struct expression *copy;
                struct symbol *field;
                int offset = 0;

                copy = next_designators(old->ident_expression,
                                        old->ctype, e, v);
                if (!copy) {
                        field = old->field->next_subobject;
                        if (!field) {
                                convert_ident(old);
                                return NULL;
                        }
                        copy = e;
                        *v = new = alloc_expression(e->pos, EXPR_IDENTIFIER);
                        /*
                         * We can't necessarily trust "field->offset",
                         * because the field might be in an anonymous
                         * union, and the field offset is then the offset
                         * within that union.
                         *
                         * The "old->offset - old->field->offset"
                         * would be the offset of such an anonymous
                         * union.
                         */
                        offset = old->offset - old->field->offset;
                } else {
                        field = old->field;
                        new = alloc_expression(e->pos, EXPR_IDENTIFIER);
                }

                new->field = field;
```

```
2503                 new->expr_ident = field->ident;
2504                 new->ident_expression = copy;
2505                 new->ctype = field;
2506                 new->offset = field->offset + offset;
2507                 convert_ident(old);
2508         }
2509         return new;
2510 }

2512 static int handle_initializer(struct expression **ep, int nested,
2513                 int class, struct symbol *ctype, unsigned long mods);

2515 /*
2516  * deal with traversing subobjects [6.7.8(17,18,20)]
2517  */
2518 static void handle_list_initializer(struct expression *expr,
2519                 int class, struct symbol *ctype, unsigned long mods)
2520 {
2521         struct expression *e, *last = NULL, *top = NULL, *next;
2522         int jumped = 0;

2524         FOR_EACH_PTR(expr->expr_list, e) {
2525                 struct expression **v;
2526                 struct symbol *type;
2527                 int lclass;

2529                 if (e->type != EXPR_INDEX && e->type != EXPR_IDENTIFIER) {
2530                         struct symbol *struct_sym;
2531                         if (!top) {
2532                                 top = e;
2533                                 last = first_subobject(ctype, class, &top);
2534                         } else {
2535                                 last = next_designators(last, ctype, e, &top);
2536                         }
2537                         if (!last) {
2538                                 excess(e, class & TYPE_PTR ? "array" :
2539                                                 "struct or union");
2540                                 DELETE_CURRENT_PTR(e);
2541                                 continue;
2542                         }
2543                         struct_sym = ctype->type == SYM_NODE ? ctype->ctype.base
2544                         if (Wdesignated_init && struct_sym->designated_init)
2545                                 warning(e->pos, "%s%.*s%spositional init of fiel
2546                                         ctype->ident ? "in initializer for " : "
2547                                         ctype->ident ? ctype->ident->len : 0,
2548                                         ctype->ident ? ctype->ident->name : "",
2549                                         ctype->ident ? ": " : "",
2550                                         get_type_name(struct_sym->type),
2551                                         show_ident(struct_sym->ident));
2552                         if (jumped) {
2553                                 warning(e->pos, "advancing past deep designator"
2554                                 jumped = 0;
2555                         }
2556                         REPLACE_CURRENT_PTR(e, last);
2557                 } else {
2558                         next = check_designators(e, ctype);
2559                         if (!next) {
2560                                 DELETE_CURRENT_PTR(e);
2561                                 continue;
2562                         }
2563                         top = next;
2564                         /* deeper than one designator? */
2565                         jumped = top != e;
2566                         convert_designators(last);
2567                         last = e;
2568                 }
```



```
2570 found:
2571                 lclass = classify_type(top->ctype, &type);
2572                 if (top->type == EXPR_INDEX)
2573                         v = &top->idx_expression;
2574                 else
2575                         v = &top->ident_expression;

2577                 mods |= ctype->ctype.modifiers & MOD_STORAGE;
2578                 if (handle_initializer(v, 1, lclass, top->ctype, mods))
2579                         continue;

2581                 if (!(lclass & TYPE_COMPOUND)) {
2582                         warning(e->pos, "bogus scalar initializer");
2583                         DELETE_CURRENT_PTR(e);
2584                         continue;
2585                 }

2587                 next = first_subobject(type, lclass, v);
2588                 if (next) {
2589                         warning(e->pos, "missing braces around initializer");
2590                         top = next;
2591                         goto found;
2592                 }

2594                 DELETE_CURRENT_PTR(e);
2595                 excess(e, lclass & TYPE_PTR ? "array" : "struct or union");

2597         } END_FOR_EACH_PTR(e);

2599         convert_designators(last);
2600         expr->ctype = ctype;
2601 }

2603 static int is_string_literal(struct expression **v)
2604 {
2605         struct expression *e = *v;
2606         while (e && e->type == EXPR_PREOP && e->op == '(')
2607                 e = e->unop;
2608         if (!e || e->type != EXPR_STRING)
2609                 return 0;
2610         if (e != *v && Wparen_string)
2611                 warning(e->pos,
2612                         "array initialized from parenthesized string constant");
2613         *v = e;
2614         return 1;
2615 }

2617 /*
2618  * We want a normal expression, possibly in one layer of braces.  Warn
2619  * if the latter happens inside a list (it's legal, but likely to be
2620  * an effect of screwup).  In case of anything not legal, we are definitely
2621  * having an effect of screwup, so just fail and let the caller warn.
2622  */
2623 static struct expression *handle_scalar(struct expression *e, int nested)
2624 {
2625         struct expression *v = NULL, *p;
2626         int count = 0;

2628         /* normal case */
2629         if (e->type != EXPR_INITIALIZER)
2630                 return e;

2632         FOR_EACH_PTR(e->expr_list, p) {
2633                 if (!v)
2634                         v = p;
```

```
2635                 count++;
2636         } END_FOR_EACH_PTR(p);
2637         if (count != 1)
2638                 return NULL;
2639         switch(v->type) {
2640         case EXPR_INITIALIZER:
2641         case EXPR_INDEX:
2642         case EXPR_IDENTIFIER:
2643                 return NULL;
2644         default:
2645                 break;
2646         }
2647         if (nested)
2648                 warning(e->pos, "braces around scalar initializer");
2649         return v;
2650 }

2652 /*
2653  * deal with the cases that don't care about subobjects:
2654  * scalar <- assignment expression, possibly in braces [6.7.8(11)]
2655  * character array <- string literal, possibly in braces [6.7.8(14)]
2656  * struct or union <- assignment expression of compatible type [6.7.8(13)]
2657  * compound type <- initializer list in braces [6.7.8(16)]
2658  * The last one punts to handle_list_initializer() which, in turn will call
2659  * us for individual elements of the list.
2660  *
2661  * We do not handle 6.7.8(15) (wide char array <- wide string literal) for
2662  * the lack of support of wide char stuff in general.
2663  *
2664  * One note: we need to take care not to evaluate a string literal until
2665  * we know that we *will* handle it right here.  Otherwise we would screw
2666  * the cases like struct { struct {char s[10]; ...} ...} initialized with
2667  * { "string", ...} - we need to preserve that string literal recognizable
2668  * until we dig into the inner struct.
2669  */
2670 static int handle_initializer(struct expression **ep, int nested,
2671                 int class, struct symbol *ctype, unsigned long mods)
2672 {
2673         int is_string = is_string_type(ctype);
2674         struct expression *e = *ep, *p;
2675         struct symbol *type;

2677         if (!e)
2678                 return 0;

2680         /* scalar */
2681         if (!(class & TYPE_COMPOUND)) {
2682                 e = handle_scalar(e, nested);
2683                 if (!e)
2684                         return 0;
2685                 *ep = e;
2686                 if (!evaluate_expression(e))
2687                         return 1;
2688                 compatible_assignment_types(e, ctype, ep, "initializer");
2689                 /*
2690                  * Initializers for static storage duration objects
2691                  * shall be constant expressions or a string literal [6.7.8(4)].
2692                  */
2693                 mods |= ctype->ctype.modifiers;
2694                 mods &= (MOD_TOPLEVEL | MOD_STATIC);
2695                 if (mods && !(e->flags & (CEF_ACE | CEF_ADDR)))
2696                         if (Wconstexpr_not_const)
2697                                 warning(e->pos, "non-constant initializer for st

2699                 return 1;
2700         }
```



```
2702         /*
2703          * sublist; either a string, or we dig in; the latter will deal with
2704          * pathologies, so we don't need anything fancy here.
2705          */
2706         if (e->type == EXPR_INITIALIZER) {
2707                 if (is_string) {
2708                         struct expression *v = NULL;
2709                         int count = 0;

2711                         FOR_EACH_PTR(e->expr_list, p) {
2712                                 if (!v)
2713                                         v = p;
2714                                 count++;
2715                         } END_FOR_EACH_PTR(p);
2716                         if (count == 1 && is_string_literal(&v)) {
2717                                 *ep = e = v;
2718                                 goto String;
2719                         }
2720                 }
2721                 handle_list_initializer(e, class, ctype, mods);
2722                 return 1;
2723         }

2725         /* string */
2726         if (is_string_literal(&e)) {
2727                 /* either we are doing array of char, or we'll have to dig in */
2728                 if (is_string) {
2729                         *ep = e;
2730                         goto String;
2731                 }
2732                 return 0;
2733         }
2734         /* struct or union can be initialized by compatible */
2735         if (class != TYPE_COMPOUND)
2736                 return 0;
2737         type = evaluate_expression(e);
2738         if (!type)
2739                 return 0;
2740         if (ctype->type == SYM_NODE)
2741                 ctype = ctype->ctype.base_type;
2742         if (type->type == SYM_NODE)
2743                 type = type->ctype.base_type;
2744         if (ctype == type)
2745                 return 1;
2746         return 0;

2748 String:
2749         p = alloc_expression(e->pos, EXPR_STRING);
2750         *p = *e;
2751         type = evaluate_expression(p);
2752         if (ctype->bit_size != -1) {
2753                 if (ctype->bit_size + bits_in_char < type->bit_size)
2754                         warning(e->pos,
2755                                 "too long initializer-string for array of char")
2756                 else if (Winit_cstring && ctype->bit_size + bits_in_char == type
2757                         warning(e->pos,
2758                                 "too long initializer-string for array of char(n
2759                 }
2760         }
2761         *ep = p;
2762         return 1;
2763 }

2765 static void evaluate_initializer(struct symbol *ctype, struct expression **ep)
2766 {
```

```
2767         struct symbol *type;
2768         int class = classify_type(ctype, &type);
2769         if (!handle_initializer(ep, 0, class, ctype, 0))
2770                 expression_error(*ep, "invalid initializer");
2771 }

2773 static struct symbol *cast_to_bool(struct expression *expr)
2774 {
2775         struct expression *old = expr->cast_expression;
2776         struct expression *zero;
2777         struct symbol *otype;
2778         int oclass = classify_type(degenerate(old), &otype);
2779         struct symbol *ctype;

2781         if (oclass & TYPE_COMPOUND)
2782                 return NULL;

2784         zero = alloc_const_expression(expr->pos, 0);
2785         expr->op = SPECIAL_NOTEQUAL;
2786         ctype = usual_conversions(expr->op, old, zero,
2787                         oclass, TYPE_NUM, otype, zero->ctype);
2788         expr->type = EXPR_COMPARE;
2789         expr->left = cast_to(old, ctype);
2790         expr->right = cast_to(zero, ctype);

2792         return expr->ctype;
2793 }

2795 static int cast_flags(struct expression *expr, struct expression *old)
2796 {
2797         struct symbol *t;
2798         int class;
2799         int flags = CEF_NONE;

2801         class = classify_type(expr->ctype, &t);
2802         if (class & TYPE_NUM) {
2803                 flags = old->flags & ~CEF_CONST_MASK;
2804                 /*
2805                  * Casts to numeric types never result in address
2806                  * constants [6.6(9)].
2807                  */
2808                 flags &= ~CEF_ADDR;

2810                 /*
2811                  * As an extension, treat address constants cast to
2812                  * integer type as an arithmetic constant.
2813                  */
2814                 if (old->flags & CEF_ADDR)
2815                         flags = CEF_ACE;

2817                 /*
2818                  * Cast to float type -> not an integer constant
2819                  * expression [6.6(6)].
2820                  */
2821                 if (class & TYPE_FLOAT)
2822                         flags &= ~CEF_CLR_ICE;
2823                 /*
2824                  * Casts of float literals to integer type results in
2825                  * a constant integer expression [6.6(6)].
2826                  */
2827                 else if (old->flags & CEF_FLOAT)
2828                         flags = CEF_SET_ICE;
2829         } else if (class & TYPE_PTR) {
2830                 /*
2831                  * Casts of integer literals to pointer type yield
2832                  * address constants [6.6(9)].
```



```
2833                  *
2834                  * As an extension, treat address constants cast to a
2835                  * different pointer type as address constants again.
2836                  *
2837                  * As another extension, treat integer constant
2838                  * expressions (in contrast to literals) cast to
2839                  * pointer type as address constants.
2840                  */
2841                 if (old->flags & (CEF_ICE | CEF_ADDR))
2842                         flags = CEF_ADDR;
2843         }

2845         return flags;
2846 }

2848 static struct symbol *evaluate_cast(struct expression *expr)
2849 {
2850         struct expression *target = expr->cast_expression;
2851         struct symbol *ctype;
2852         struct symbol *t1, *t2;
2853         int class1, class2;
2854         int as1 = 0, as2 = 0;

2856         if (!target)
2857                 return NULL;

2859         /*
2860          * Special case: a cast can be followed by an
2861          * initializer, in which case we need to pass
2862          * the type value down to that initializer rather
2863          * than trying to evaluate it as an expression
2864          *
2865          * A more complex case is when the initializer is
2866          * dereferenced as part of a post-fix expression.
2867          * We need to produce an expression that can be dereferenced.
2868          */
2869         if (target->type == EXPR_INITIALIZER) {
2870                 struct symbol *sym = expr->cast_type;
2871                 struct expression *addr = alloc_expression(expr->pos, EXPR_SYMBO

2873                 sym->initializer = target;
2874                 evaluate_symbol(sym);

2876                 addr->ctype = &lazy_ptr_ctype;  /* Lazy eval */
2877                 addr->symbol = sym;
2878                 if (sym->ctype.modifiers & MOD_TOPLEVEL)
2879                         addr->flags |= CEF_ADDR;

2881                 expr->type = EXPR_PREOP;
2882                 expr->op = '*';
2883                 expr->unop = addr;
2884                 expr->ctype = sym;

2886                 return sym;
2887         }

2889         ctype = examine_symbol_type(expr->cast_type);
2890         expr->ctype = ctype;
2891         expr->cast_type = ctype;

2893         evaluate_expression(target);
2894         degenerate(target);

2896         class1 = classify_type(ctype, &t1);

2898         expr->flags = cast_flags(expr, target);
```

```
2900         /*
2901          * You can always throw a value away by casting to
2902          * "void" - that's an implicit "force". Note that
2903          * the same is _not_ true of "void *".
2904          */
2905         if (t1 == &void_ctype)
2906                 goto out;

2908         if (class1 & (TYPE_COMPOUND | TYPE_FN))
2909                 warning(expr->pos, "cast to non-scalar");

2911         t2 = target->ctype;
2912         if (!t2) {
2913                 expression_error(expr, "cast from unknown type");
2914                 goto out;
2915         }
2916         class2 = classify_type(t2, &t2);

2918         if (class2 & TYPE_COMPOUND)
2919                 warning(expr->pos, "cast from non-scalar");

2921         if (expr->type == EXPR_FORCE_CAST)
2922                 goto out;

2924         /* allowed cast unfouls */
2925         if (class2 & TYPE_FOULED)
2926                 t2 = unfoul(t2);

2928         if (t1 != t2) {
2929                 if ((class1 & TYPE_RESTRICT) && restricted_value(target, t1))
2930                         warning(expr->pos, "cast to %s",
2931                                 show_typename(t1));
2932                 if (class2 & TYPE_RESTRICT) {
2933                         if (t1 == &bool_ctype) {
2934                                 if (class2 & TYPE_FOULED)
2935                                         warning(expr->pos, "%s degrades to integ
2936                                                 show_typename(t2));
2937                         } else {
2938                                 warning(expr->pos, "cast from %s",
2939                                         show_typename(t2));
2940                         }
2941                 }
2942         }

2944         if (t1 == &ulong_ctype)
2945                 as1 = -1;
2946         else if (class1 == TYPE_PTR) {
2947                 examine_pointer_target(t1);
2948                 as1 = t1->ctype.as;
2949         }

2951         if (t2 == &ulong_ctype)
2952                 as2 = -1;
2953         else if (class2 == TYPE_PTR) {
2954                 examine_pointer_target(t2);
2955                 as2 = t2->ctype.as;
2956         }

2958         if (!as1 && as2 > 0)
2959                 warning(expr->pos, "cast removes address space of expression");
2960         if (as1 > 0 && as2 > 0 && as1 != as2)
2961                 warning(expr->pos, "cast between address spaces (<asn:%d>-><asn:
2962         if (as1 > 0 && !as2 &&
2963             !is_null_pointer_constant(target) && Wcast_to_as)
2964                 warning(expr->pos,
```



```
2965                         "cast adds address space to expression (<asn:%d>)", as1)

2967         if (!(t1->ctype.modifiers & MOD_PTRINHERIT) && class1 == TYPE_PTR &&
2968             !as1 && (target->flags & CEF_ICE)) {
2969                 if (t1->ctype.base_type == &void_ctype) {
2970                         if (is_zero_constant(target)) {
2971                                 /* NULL */
2972                                 expr->type = EXPR_VALUE;
2973                                 expr->ctype = &null_ctype;
2974                                 expr->value = 0;
2975                                 return expr->ctype;
2976                         }
2977                 }
2978         }

2980         if (t1 == &bool_ctype)
2981                 cast_to_bool(expr);

2983 out:
2984         return ctype;
2985 }

2987 /*
2988  * Evaluate a call expression with a symbol. This
2989  * should expand inline functions, and evaluate
2990  * builtins.
2991  */
2992 static int evaluate_symbol_call(struct expression *expr)
2993 {
2994         struct expression *fn = expr->fn;
2995         struct symbol *ctype = fn->ctype;

2997         if (fn->type != EXPR_PREOP)
2998                 return 0;

3000         if (ctype->op && ctype->op->evaluate)
3001                 return ctype->op->evaluate(expr);

3003         if (ctype->ctype.modifiers & MOD_INLINE) {
3004                 int ret;
3005                 struct symbol *curr = current_fn;

3007                 if (ctype->definition)
3008                         ctype = ctype->definition;

3010                 current_fn = ctype->ctype.base_type;

3012                 ret = inline_function(expr, ctype);

3014                 /* restore the old function */
3015                 current_fn = curr;
3016                 return ret;
3017         }

3019         return 0;
3020 }

3022 static struct symbol *evaluate_call(struct expression *expr)
3023 {
3024         int args, fnargs;
3025         struct symbol *ctype, *sym;
3026         struct expression *fn = expr->fn;
3027         struct expression_list *arglist = expr->args;

3029         if (!evaluate_expression(fn))
3030                 return NULL;
```

```
3031         sym = ctype = fn->ctype;
3032         if (ctype->type == SYM_NODE)
3033                 ctype = ctype->ctype.base_type;
3034         if (ctype->type == SYM_PTR)
3035                 ctype = get_base_type(ctype);

3037         if (ctype->type != SYM_FN) {
3038                 struct expression *arg;
3039                 expression_error(expr, "not a function %s",
3040                                  show_ident(sym->ident));
3041                 /* do typechecking in arguments */
3042                 FOR_EACH_PTR (arglist, arg) {
3043                         evaluate_expression(arg);
3044                 } END_FOR_EACH_PTR(arg);
3045                 return NULL;
3046         }

3048         examine_fn_arguments(ctype);
3049         if (sym->type == SYM_NODE && fn->type == EXPR_PREOP &&
3050             sym->op && sym->op->args) {
3051                 if (!sym->op->args(expr))
3052                         return NULL;
3053         } else {
3054                 if (!evaluate_arguments(ctype, arglist))
3055                         return NULL;
3056                 args = expression_list_size(expr->args);
3057                 fnargs = symbol_list_size(ctype->arguments);
3058                 if (args < fnargs) {
3059                         expression_error(expr,
3060                                          "not enough arguments for function %s",
3061                                          show_ident(sym->ident));
3062                         return NULL;
3063                 }
3064                 if (args > fnargs && !ctype->variadic)
3065                         expression_error(expr,
3066                                          "too many arguments for function %s",
3067                                          show_ident(sym->ident));
3068         }
3069         expr->ctype = ctype->ctype.base_type;
3070         if (sym->type == SYM_NODE) {
3071                 if (evaluate_symbol_call(expr))
3072                         return expr->ctype;
3073         }
3074         return expr->ctype;
3075 }

3077 static struct symbol *evaluate_offsetof(struct expression *expr)
3078 {
3079         struct expression *e = expr->down;
3080         struct symbol *ctype = expr->in;
3081         int class;

3083         if (expr->op == '.') {
3084                 struct symbol *field;
3085                 int offset = 0;
3086                 if (!ctype) {
3087                         expression_error(expr, "expected structure or union");
3088                         return NULL;
3089                 }
3090                 examine_symbol_type(ctype);
3091                 class = classify_type(ctype, &ctype);
3092                 if (class != TYPE_COMPOUND) {
3093                         expression_error(expr, "expected structure or union");
3094                         return NULL;
3095                 }
```



```
3097                 field = find_identifier(expr->ident, ctype->symbol_list, &offset
3098                 if (!field) {
3099                         expression_error(expr, "unknown member");
3100                         return NULL;
3101                 }
3102                 ctype = field;
3103                 expr->type = EXPR_VALUE;
3104                 expr->flags = CEF_SET_ICE;
3105                 expr->value = offset;
3106                 expr->taint = 0;
3107                 expr->ctype = size_t_ctype;
3108         } else {
3109                 if (!ctype) {
3110                         expression_error(expr, "expected structure or union");
3111                         return NULL;
3112                 }
3113                 examine_symbol_type(ctype);
3114                 class = classify_type(ctype, &ctype);
3115                 if (class != (TYPE_COMPOUND | TYPE_PTR)) {
3116                         expression_error(expr, "expected array");
3117                         return NULL;
3118                 }
3119                 ctype = ctype->ctype.base_type;
3120                 if (!expr->index) {
3121                         expr->type = EXPR_VALUE;
3122                         expr->flags = CEF_SET_ICE;
3123                         expr->value = 0;
3124                         expr->taint = 0;
3125                         expr->ctype = size_t_ctype;
3126                 } else {
3127                         struct expression *idx = expr->index, *m;
3128                         struct symbol *i_type = evaluate_expression(idx);
3129                         unsigned old_idx_flags;
3130                         int i_class = classify_type(i_type, &i_type);

3132                         if (!is_int(i_class)) {
3133                                 expression_error(expr, "non-integer index");
3134                                 return NULL;
3135                         }
3136                         unrestrict(idx, i_class, &i_type);
3137                         old_idx_flags = idx->flags;
3138                         idx = cast_to(idx, size_t_ctype);
3139                         idx->flags = old_idx_flags;
3140                         m = alloc_const_expression(expr->pos,
3141                                                    bits_to_bytes(ctype->bit_size
3142                         m->ctype = size_t_ctype;
3143                         m->flags = CEF_SET_INT;
3144                         expr->type = EXPR_BINOP;
3145                         expr->left = idx;
3146                         expr->right = m;
3147                         expr->op = '*';
3148                         expr->ctype = size_t_ctype;
3149                         expr->flags = m->flags & idx->flags & ~CEF_CONST_MASK;
3150                 }
3151         }
3152         if (e) {
3153                 struct expression *copy = __alloc_expression(0);
3154                 *copy = *expr;
3155                 if (e->type == EXPR_OFFSETOF)
3156                         e->in = ctype;
3157                 if (!evaluate_expression(e))
3158                         return NULL;
3159                 expr->type = EXPR_BINOP;
3160                 expr->flags = e->flags & copy->flags & ~CEF_CONST_MASK;
3161                 expr->op = '+';
3162                 expr->ctype = size_t_ctype;
```

```
3163                 expr->left = copy;
3164                 expr->right = e;
3165         }
3166         return size_t_ctype;
3167 }

3169 struct symbol *evaluate_expression(struct expression *expr)
3170 {
3171         if (!expr)
3172                 return NULL;
3173         if (expr->ctype)
3174                 return expr->ctype;

3176         switch (expr->type) {
3177         case EXPR_VALUE:
3178         case EXPR_FVALUE:
3179                 expression_error(expr, "value expression without a type");
3180                 return NULL;
3181         case EXPR_STRING:
3182                 return evaluate_string(expr);
3183         case EXPR_SYMBOL:
3184                 return evaluate_symbol_expression(expr);
3185         case EXPR_BINOP:
3186                 if (!evaluate_expression(expr->left))
3187                         return NULL;
3188                 if (!evaluate_expression(expr->right))
3189                         return NULL;
3190                 return evaluate_binop(expr);
3191         case EXPR_LOGICAL:
3192                 return evaluate_logical(expr);
3193         case EXPR_COMMA:
3194                 evaluate_expression(expr->left);
3195                 if (!evaluate_expression(expr->right))
3196                         return NULL;
3197                 return evaluate_comma(expr);
3198         case EXPR_COMPARE:
3199                 if (!evaluate_expression(expr->left))
3200                         return NULL;
3201                 if (!evaluate_expression(expr->right))
3202                         return NULL;
3203                 return evaluate_compare(expr);
3204         case EXPR_ASSIGNMENT:
3205                 if (!evaluate_expression(expr->left))
3206                         return NULL;
3207                 if (!evaluate_expression(expr->right))
3208                         return NULL;
3209                 return evaluate_assignment(expr);
3210         case EXPR_PREOP:
3211                 if (!evaluate_expression(expr->unop))
3212                         return NULL;
3213                 return evaluate_preop(expr);
3214         case EXPR_POSTOP:
3215                 if (!evaluate_expression(expr->unop))
3216                         return NULL;
3217                 return evaluate_postop(expr);
3218         case EXPR_CAST:
3219         case EXPR_FORCE_CAST:
3220         case EXPR_IMPLIED_CAST:
3221                 return evaluate_cast(expr);
3222         case EXPR_SIZEOF:
3223                 return evaluate_sizeof(expr);
3224         case EXPR_PTRSIZEOF:
3225                 return evaluate_ptrsizeof(expr);
3226         case EXPR_ALIGNOF:
3227                 return evaluate_alignof(expr);
3228         case EXPR_DEREF:
```



```
3229                 return evaluate_member_dereference(expr);
3230         case EXPR_CALL:
3231                 return evaluate_call(expr);
3232         case EXPR_SELECT:
3233         case EXPR_CONDITIONAL:
3234                 return evaluate_conditional_expression(expr);
3235         case EXPR_STATEMENT:
3236                 expr->ctype = evaluate_statement(expr->statement);
3237                 return expr->ctype;

3239         case EXPR_LABEL:
3240                 expr->ctype = &ptr_ctype;
3241                 return &ptr_ctype;

3243         case EXPR_TYPE:
3244                 /* Evaluate the type of the symbol .. */
3245                 evaluate_symbol(expr->symbol);
3246                 /* .. but the type of the _expression_ is a "type" */
3247                 expr->ctype = &type_ctype;
3248                 return &type_ctype;

3250         case EXPR_OFFSETOF:
3251                 return evaluate_offsetof(expr);

3253         /* These can not exist as stand-alone expressions */
3254         case EXPR_INITIALIZER:
3255         case EXPR_IDENTIFIER:
3256         case EXPR_INDEX:
3257         case EXPR_POS:
3258                 expression_error(expr, "internal front-end error: initializer in
3259                 return NULL;
3260         case EXPR_SLICE:
3261                 expression_error(expr, "internal front-end error: SLICE re-evalu
3262                 return NULL;
3263         }
3264         return NULL;
3265 }

3267 static void check_duplicates(struct symbol *sym)
3268 {
3269         int declared = 0;
3270         struct symbol *next = sym;
3271         int initialized = sym->initializer != NULL;

3273         while ((next = next->same_symbol) != NULL) {
3274                 const char *typediff;
3275                 evaluate_symbol(next);
3276                 if (initialized && next->initializer) {
3277                         sparse_error(sym->pos, "symbol '%s' has multiple initial
3278                                 show_ident(sym->ident),
3279                                 stream_name(next->pos.stream), next->pos.line);
3280                         /* Only warn once */
3281                         initialized = 0;
3282                 }
3283                 declared++;
3284                 typediff = type_difference(&sym->ctype, &next->ctype, 0, 0);
3285                 if (typediff) {
3286                         sparse_error(sym->pos, "symbol '%s' redeclared with diff
3287                                 show_ident(sym->ident),
3288                                 stream_name(next->pos.stream), next->pos.line, t
3289                         return;
3290                 }
3291         }
3292         if (!declared) {
3293                 unsigned long mod = sym->ctype.modifiers;
3294                 if (mod & (MOD_STATIC | MOD_REGISTER))
```

```
3295                            return;
3296                    if (!(mod & MOD_TOPLEVEL))
3297                            return;
3298                    if (!Wdecl)
3299                            return;
3300                    if (sym->ident == &main_ident)
3301                            return;
3302                    warning(sym->pos, "symbol '%s' was not declared. Should it be st
3303            }
3304    }

3306    static struct symbol *evaluate_symbol(struct symbol *sym)
3307    {
3308            struct symbol *base_type;

3310            if (!sym)
3311                    return sym;
3312            if (sym->evaluated)
3313                    return sym;
3314            sym->evaluated = 1;

3316            sym = examine_symbol_type(sym);
3317            base_type = get_base_type(sym);
3318            if (!base_type)
3319                    return NULL;

3321            /* Evaluate the initializers */
3322            if (sym->initializer)
3323                    evaluate_initializer(sym, &sym->initializer);

3325            /* And finally, evaluate the body of the symbol too */
3326            if (base_type->type == SYM_FN) {
3327                    struct symbol *curr = current_fn;

3329                    if (sym->definition && sym->definition != sym)
3330                            return evaluate_symbol(sym->definition);

3332                    current_fn = base_type;

3334                    examine_fn_arguments(base_type);
3335                    if (!base_type->stmt && base_type->inline_stmt)
3336                            uninline(sym);
3337                    if (base_type->stmt)
3338                            evaluate_statement(base_type->stmt);

3340                    current_fn = curr;
3341            }

3343            return base_type;
3344    }

3346    void evaluate_symbol_list(struct symbol_list *list)
3347    {
3348            struct symbol *sym;

3350            FOR_EACH_PTR(list, sym) {
3351                    has_error &= ~ERROR_CURR_PHASE;
3352                    evaluate_symbol(sym);
3353                    check_duplicates(sym);
3354            } END_FOR_EACH_PTR(sym);
3355    }

3357    static struct symbol *evaluate_return_expression(struct statement *stmt)
3358    {
3359            struct expression *expr = stmt->expression;
3360            struct symbol *fntype;
```



```
3362            evaluate_expression(expr);
3363            fntype = current_fn->ctype.base_type;
3364            if (!fntype || fntype == &void_ctype) {
3365                    if (expr && expr->ctype != &void_ctype)
3366                            expression_error(expr, "return expression in %s function
3367                    if (expr && Wreturn_void)
3368                            warning(stmt->pos, "returning void-valued expression");
3369                    return NULL;
3370            }

3372            if (!expr) {
3373                    sparse_error(stmt->pos, "return with no return value");
3374                    return NULL;
3375            }
3376            if (!expr->ctype)
3377                    return NULL;
3378            compatible_assignment_types(expr, fntype, &stmt->expression, "return exp
3379            return NULL;
3380    }

3382    static void evaluate_if_statement(struct statement *stmt)
3383    {
3384            if (!stmt->if_conditional)
3385                    return;

3387            evaluate_conditional(stmt->if_conditional, 0);
3388            evaluate_statement(stmt->if_true);
3389            evaluate_statement(stmt->if_false);
3390    }

3392    static void evaluate_iterator(struct statement *stmt)
3393    {
3394            evaluate_symbol_list(stmt->iterator_syms);
3395            evaluate_conditional(stmt->iterator_pre_condition, 1);
3396            evaluate_conditional(stmt->iterator_post_condition,1);
3397            evaluate_statement(stmt->iterator_pre_statement);
3398            evaluate_statement(stmt->iterator_statement);
3399            evaluate_statement(stmt->iterator_post_statement);
3400    }

3402    static void verify_output_constraint(struct expression *expr, const char *constr
3403    {
3404            switch (*constraint) {
3405            case '=':       /* Assignment */
3406            case '+':       /* Update */
3407                    break;
3408            default:
3409                    expression_error(expr, "output constraint is not an assignment c
3410            }
3411    }

3413    static void verify_input_constraint(struct expression *expr, const char *constra
3414    {
3415            switch (*constraint) {
3416            case '=':       /* Assignment */
3417            case '+':       /* Update */
3418                    expression_error(expr, "input constraint with assignment (\"%s\"
3419            }
3420    }

3422    static void evaluate_asm_statement(struct statement *stmt)
3423    {
3424            struct expression *expr;
3425            struct symbol *sym;
3426            int state;
```

```
3428         expr = stmt->asm_string;
3429         if (!expr || expr->type != EXPR_STRING) {
3430                 sparse_error(stmt->pos, "need constant string for inline asm");
3431                 return;
3432         }
3433
3434         state = 0;
3435         FOR_EACH_PTR(stmt->asm_outputs, expr) {
3436                 switch (state) {
3437                 case 0: /* Identifier */
3438                         state = 1;
3439                         continue;
3440
3441                 case 1: /* Constraint */
3442                         state = 2;
3443                         if (!expr || expr->type != EXPR_STRING) {
3444                                 sparse_error(expr ? expr->pos : stmt->pos, "asm
3445                                 *THIS_ADDRESS(expr) = NULL;
3446                                 continue;
3447                         }
3448                         verify_output_constraint(expr, expr->string->data);
3449                         continue;
3450
3451                 case 2: /* Expression */
3452                         state = 0;
3453                         if (!evaluate_expression(expr))
3454                                 return;
3455                         if (!lvalue_expression(expr))
3456                                 warning(expr->pos, "asm output is not an lvalue"
3457                         evaluate_assign_to(expr, expr->ctype);
3458                         continue;
3459                 }
3460         } END_FOR_EACH_PTR(expr);
3461
3462         state = 0;
3463         FOR_EACH_PTR(stmt->asm_inputs, expr) {
3464                 switch (state) {
3465                 case 0: /* Identifier */
3466                         state = 1;
3467                         continue;
3468
3469                 case 1: /* Constraint */
3470                         state = 2;
3471                         if (!expr || expr->type != EXPR_STRING) {
3472                                 sparse_error(expr ? expr->pos : stmt->pos, "asm
3473                                 *THIS_ADDRESS(expr) = NULL;
3474                                 continue;
3475                         }
3476                         verify_input_constraint(expr, expr->string->data);
3477                         continue;
3478
3479                 case 2: /* Expression */
3480                         state = 0;
3481                         if (!evaluate_expression(expr))
3482                                 return;
3483                         continue;
3484                 }
3485         } END_FOR_EACH_PTR(expr);
3486
3487         FOR_EACH_PTR(stmt->asm_clobbers, expr) {
3488                 if (!expr) {
3489                         sparse_error(stmt->pos, "bad asm clobbers");
3490                         return;
3491                 }
3492                 if (expr->type == EXPR_STRING)
```



```
3493                         continue;
3494                 expression_error(expr, "asm clobber is not a string");
3495         } END_FOR_EACH_PTR(expr);
3496
3497         FOR_EACH_PTR(stmt->asm_labels, sym) {
3498                 if (!sym || sym->type != SYM_LABEL) {
3499                         sparse_error(stmt->pos, "bad asm label");
3500                         return;
3501                 }
3502         } END_FOR_EACH_PTR(sym);
3503 }
3504
3505 static void evaluate_case_statement(struct statement *stmt)
3506 {
3507         evaluate_expression(stmt->case_expression);
3508         evaluate_expression(stmt->case_to);
3509         evaluate_statement(stmt->case_statement);
3510 }
3511
3512 static void check_case_type(struct expression *switch_expr,
3513                             struct expression *case_expr,
3514                             struct expression **enumcase)
3515 {
3516         struct symbol *switch_type, *case_type;
3517         int sclass, cclass;
3518
3519         if (!case_expr)
3520                 return;
3521
3522         switch_type = switch_expr->ctype;
3523         case_type = evaluate_expression(case_expr);
3524
3525         if (!switch_type || !case_type)
3526                 goto Bad;
3527         if (enumcase) {
3528                 if (*enumcase)
3529                         warn_for_different_enum_types(case_expr->pos, case_type,
3530                 else if (is_enum_type(case_type))
3531                         *enumcase = case_expr;
3532         }
3533
3534         sclass = classify_type(switch_type, &switch_type);
3535         cclass = classify_type(case_type, &case_type);
3536
3537         /* both should be arithmetic */
3538         if (!(sclass & cclass & TYPE_NUM))
3539                 goto Bad;
3540
3541         /* neither should be floating */
3542         if ((sclass | cclass) & TYPE_FLOAT)
3543                 goto Bad;
3544
3545         /* if neither is restricted, we are OK */
3546         if (!((sclass | cclass) & TYPE_RESTRICT))
3547                 return;
3548
3549         if (!restricted_binop_type(SPECIAL_EQUAL, case_expr, switch_expr,
3550                                    cclass, sclass, case_type, switch_type)) {
3551                 unrestrict(case_expr, cclass, &case_type);
3552                 unrestrict(switch_expr, sclass, &switch_type);
3553         }
3554         return;
3555
3556 Bad:
3557         expression_error(case_expr, "incompatible types for ‘case’ statement");
3558 }
```

```
3560 static void evaluate_switch_statement(struct statement *stmt)
3561 {
3562         struct symbol *sym;
3563         struct expression *enumcase = NULL;
3564         struct expression **enumcase_holder = &enumcase;
3565         struct expression *sel = stmt->switch_expression;

3567         evaluate_expression(sel);
3568         evaluate_statement(stmt->switch_statement);
3569         if (!sel)
3570                 return;
3571         if (sel->ctype && is_enum_type(sel->ctype))
3572                 enumcase_holder = NULL; /* Only check cases against switch */

3574         FOR_EACH_PTR(stmt->switch_case->symbol_list, sym) {
3575                 struct statement *case_stmt = sym->stmt;
3576                 check_case_type(sel, case_stmt->case_expression, enumcase_holder
3577                 check_case_type(sel, case_stmt->case_to, enumcase_holder);
3578         } END_FOR_EACH_PTR(sym);
3579 }

3581 static void evaluate_goto_statement(struct statement *stmt)
3582 {
3583         struct symbol *label = stmt->goto_label;

3585         if (label && !label->stmt && !lookup_keyword(label->ident, NS_KEYWORD))
3586                 sparse_error(stmt->pos, "label '%s' was not declared", show_iden

3588         evaluate_expression(stmt->goto_expression);
3589 }

3591 struct symbol *evaluate_statement(struct statement *stmt)
3592 {
3593         if (!stmt)
3594                 return NULL;

3596         switch (stmt->type) {
3597         case STMT_DECLARATION: {
3598                 struct symbol *s;
3599                 FOR_EACH_PTR(stmt->declaration, s) {
3600                         evaluate_symbol(s);
3601                 } END_FOR_EACH_PTR(s);
3602                 return NULL;
3603         }

3605         case STMT_RETURN:
3606                 return evaluate_return_expression(stmt);

3608         case STMT_EXPRESSION:
3609                 if (!evaluate_expression(stmt->expression))
3610                         return NULL;
3611                 if (stmt->expression->ctype == &null_ctype)
3612                         stmt->expression = cast_to(stmt->expression, &ptr_ctype)
3613                 return degenerate(stmt->expression);

3615         case STMT_COMPOUND: {
3616                 struct statement *s;
3617                 struct symbol *type = NULL;

3619                 /* Evaluate the return symbol in the compound statement */
3620                 evaluate_symbol(stmt->ret);

3622                 /*
3623                  * Then, evaluate each statement, making the type of the
3624                  * compound statement be the type of the last statement
```



```
3625                  */
3626                 type = evaluate_statement(stmt->args);
3627                 FOR_EACH_PTR(stmt->stmts, s) {
3628                         type = evaluate_statement(s);
3629                 } END_FOR_EACH_PTR(s);
3630                 if (!type)
3631                         type = &void_ctype;
3632                 return type;
3633         }
3634         case STMT_IF:
3635                 evaluate_if_statement(stmt);
3636                 return NULL;
3637         case STMT_ITERATOR:
3638                 evaluate_iterator(stmt);
3639                 return NULL;
3640         case STMT_SWITCH:
3641                 evaluate_switch_statement(stmt);
3642                 return NULL;
3643         case STMT_CASE:
3644                 evaluate_case_statement(stmt);
3645                 return NULL;
3646         case STMT_LABEL:
3647                 return evaluate_statement(stmt->label_statement);
3648         case STMT_GOTO:
3649                 evaluate_goto_statement(stmt);
3650                 return NULL;
3651         case STMT_NONE:
3652                 break;
3653         case STMT_ASM:
3654                 evaluate_asm_statement(stmt);
3655                 return NULL;
3656         case STMT_CONTEXT:
3657                 evaluate_expression(stmt->expression);
3658                 return NULL;
3659         case STMT_RANGE:
3660                 evaluate_expression(stmt->range_expression);
3661                 evaluate_expression(stmt->range_low);
3662                 evaluate_expression(stmt->range_high);
3663                 return NULL;
3664         }
3665         return NULL;
3666 }
```

```
**********************************************************
   45166 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/example.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 /*
  2  * Example of how to write a compiler with sparse
  3  */
  4 #include <stdio.h>
  5 #include <stdlib.h>
  6 #include <stdarg.h>
  7 #include <string.h>
  8 #include <assert.h>

 10 #include "symbol.h"
 11 #include "expression.h"
 12 #include "linearize.h"
 13 #include "flow.h"
 14 #include "storage.h"
 15 #include "target.h"

 17 static const char *opcodes[] = {
 18         [OP_BADOP] = "bad_op",

 20         /* Fn entrypoint */
 21         [OP_ENTRY] = "<entry-point>",

 23         /* Terminator */
 24         [OP_RET] = "ret",
 25         [OP_BR] = "br",
 26         [OP_CBR] = "cbr",
 27         [OP_SWITCH] = "switch",
 28         [OP_INVOKE] = "invoke",
 29         [OP_COMPUTEDGOTO] = "jmp *",
 30         [OP_UNWIND] = "unwind",
 31
 32         /* Binary */
 33         [OP_ADD] = "add",
 34         [OP_SUB] = "sub",
 35         [OP_MULU] = "mulu",
 36         [OP_MULS] = "muls",
 37         [OP_DIVU] = "divu",
 38         [OP_DIVS] = "divs",
 39         [OP_MODU] = "modu",
 40         [OP_MODS] = "mods",
 41         [OP_SHL] = "shl",
 42         [OP_LSR] = "lsr",
 43         [OP_ASR] = "asr",
 44
 45         /* Logical */
 46         [OP_AND] = "and",
 47         [OP_OR] = "or",
 48         [OP_XOR] = "xor",
 49         [OP_AND_BOOL] = "and-bool",
 50         [OP_OR_BOOL] = "or-bool",

 52         /* Binary comparison */
 53         [OP_SET_EQ] = "seteq",
 54         [OP_SET_NE] = "setne",
 55         [OP_SET_LE] = "setle",
 56         [OP_SET_GE] = "setge",
 57         [OP_SET_LT] = "setlt",
 58         [OP_SET_GT] = "setgt",
 59         [OP_SET_B] = "setb",
 60         [OP_SET_A] = "seta",
```



```
 61         [OP_SET_BE] = "setbe",
 62         [OP_SET_AE] = "setae",

 64         /* Uni */
 65         [OP_NOT] = "not",
 66         [OP_NEG] = "neg",

 68         /* Special three-input */
 69         [OP_SEL] = "select",
 70
 71         /* Memory */
 72         [OP_MALLOC] = "malloc",
 73         [OP_FREE] = "free",
 74         [OP_ALLOCA] = "alloca",
 75         [OP_LOAD] = "load",
 76         [OP_STORE] = "store",
 77         [OP_SETVAL] = "set",
 78         [OP_GET_ELEMENT_PTR] = "getelem",

 80         /* Other */
 81         [OP_PHI] = "phi",
 82         [OP_PHISOURCE] = "phisrc",
 83         [OP_COPY] = "copy",
 84         [OP_CAST] = "cast",
 85         [OP_SCAST] = "scast",
 86         [OP_FPCAST] = "fpcast",
 87         [OP_PTRCAST] = "ptrcast",
 88         [OP_CALL] = "call",
 89         [OP_VANEXT] = "va_next",
 90         [OP_VAARG] = "va_arg",
 91         [OP_SLICE] = "slice",
 92         [OP_SNOP] = "snop",
 93         [OP_LNOP] = "lnop",
 94         [OP_NOP] = "nop",
 95         [OP_DEATHNOTE] = "dead",
 96         [OP_ASM] = "asm",

 98         /* Sparse tagging (line numbers, context, whatever) */
 99         [OP_CONTEXT] = "context",
100 };

102 static int last_reg, stack_offset;

104 struct hardreg {
105         const char *name;
106         struct pseudo_list *contains;
107         unsigned busy:16,
108                  dead:8,
109                  used:1;
110 };

112 #define TAG_DEAD 1
113 #define TAG_DIRTY 2

115 /* Our "switch" generation is very very stupid. */
116 #define SWITCH_REG (1)

118 static void output_bb(struct basic_block *bb, unsigned long generation);

120 /*
121  * We only know about the caller-clobbered registers
122  * right now.
123  */
124 static struct hardreg hardregs[] = {
125         { .name = "%eax" },
126         { .name = "%edx" },
```

```
127         { .name = "%ecx" },
128         { .name = "%ebx" },
129         { .name = "%esi" },
130         { .name = "%edi" },

132         { .name = "%ebp" },
133         { .name = "%esp" },
134 };
135 #define REGNO 6
136 #define REG_EBP 6
137 #define REG_ESP 7

139 struct bb_state {
140         struct position pos;
141         struct storage_hash_list *inputs;
142         struct storage_hash_list *outputs;
143         struct storage_hash_list *internal;

145         /* CC cache.. */
146         int cc_opcode, cc_dead;
147         pseudo_t cc_target;
148 };

150 enum optype {
151         OP_UNDEF,
152         OP_REG,
153         OP_VAL,
154         OP_MEM,
155         OP_ADDR,
156 };

158 struct operand {
159         enum optype type;
160         int size;
161         union {
162                 struct hardreg *reg;
163                 long long value;
164                 struct /* OP_MEM and OP_ADDR */ {
165                         unsigned int offset;
166                         unsigned int scale;
167                         struct symbol *sym;
168                         struct hardreg *base;
169                         struct hardreg *index;
170                 };
171         };
172 };

174 static const char *show_op(struct bb_state *state, struct operand *op)
175 {
176         static char buf[256][4];
177         static int bufnr;
178         char *p, *ret;
179         int nr;

181         nr = (bufnr + 1) & 3;
182         bufnr = nr;
183         ret = p = buf[nr];

185         switch (op->type) {
186         case OP_UNDEF:
187                 return "undef";
188         case OP_REG:
189                 return op->reg->name;
190         case OP_VAL:
191                 sprintf(p, "$%lld", op->value);
192                 break;
```



```
193         case OP_MEM:
194         case OP_ADDR:
195                 if (op->offset)
196                         p += sprintf(p, "%d", op->offset);
197                 if (op->sym)
198                         p += sprintf(p, "%s%s",
199                                 op->offset ? "+" : "",
200                                 show_ident(op->sym->ident));
201                 if (op->base || op->index) {
202                         p += sprintf(p, "(%s%s%s",
203                                 op->base ? op->base->name : "",
204                                 (op->base && op->index) ? "," : "",
205                                 op->index ? op->index->name : "");
206                         if (op->scale > 1)
207                                 p += sprintf(p, ",%d", op->scale);
208                         *p++ = ')';
209                         *p = '\0';
210                 }
211                 break;
212         }
213         return ret;
214 }

216 static struct storage_hash *find_storage_hash(pseudo_t pseudo, struct storage_ha
217 {
218         struct storage_hash *entry;
219         FOR_EACH_PTR(list, entry) {
220                 if (entry->pseudo == pseudo)
221                         return entry;
222         } END_FOR_EACH_PTR(entry);
223         return NULL;
224 }

226 static struct storage_hash *find_or_create_hash(pseudo_t pseudo, struct storage_
227 {
228         struct storage_hash *entry;

230         entry = find_storage_hash(pseudo, *listp);
231         if (!entry) {
232                 entry = alloc_storage_hash(alloc_storage());
233                 entry->pseudo = pseudo;
234                 add_ptr_list(listp, entry);
235         }
236         return entry;
237 }

239 /* Eventually we should just build it up in memory */
240 static void FORMAT_ATTR(2) output_line(struct bb_state *state, const char *fmt,
241 {
242         va_list args;

244         va_start(args, fmt);
245         vprintf(fmt, args);
246         va_end(args);
247 }

249 static void FORMAT_ATTR(2) output_label(struct bb_state *state, const char *fmt,
250 {
251         static char buffer[512];
252         va_list args;

254         va_start(args, fmt);
255         vsnprintf(buffer, sizeof(buffer), fmt, args);
256         va_end(args);

258         output_line(state, "%s:\n", buffer);
```

```
}

static void FORMAT_ATTR(2) output_insn(struct bb_state *state, const char *fmt,
{
        static char buffer[512];
        va_list args;

        va_start(args, fmt);
        vsnprintf(buffer, sizeof(buffer), fmt, args);
        va_end(args);

        output_line(state, "\t%s\n", buffer);
}

#define output_insn(state, fmt, arg...) \
        output_insn(state, fmt "\t\t# %s" , ## arg , __FUNCTION__)

static void FORMAT_ATTR(2) output_comment(struct bb_state *state, const char *fm
{
        static char buffer[512];
        va_list args;

        if (!verbose)
                return;
        va_start(args, fmt);
        vsnprintf(buffer, sizeof(buffer), fmt, args);
        va_end(args);

        output_line(state, "\t# %s\n", buffer);
}

static const char *show_memop(struct storage *storage)
{
        static char buffer[1000];

        if (!storage)
                return "undef";
        switch (storage->type) {
        case REG_FRAME:
                sprintf(buffer, "%d(FP)", storage->offset);
                break;
        case REG_STACK:
                sprintf(buffer, "%d(SP)", storage->offset);
                break;
        case REG_REG:
                return hardregs[storage->regno].name;
        default:
                return show_storage(storage);
        }
        return buffer;
}

static int alloc_stack_offset(int size)
{
        int ret = stack_offset;
        stack_offset = ret + size;
        return ret;
}

static void alloc_stack(struct bb_state *state, struct storage *storage)
{
        storage->type = REG_STACK;
        storage->offset = alloc_stack_offset(4);
}

/*
```



```
 * Can we re-generate the pseudo, so that we don't need to
 * flush it to memory? We can regenerate:
 *  - immediates and symbol addresses
 *  - pseudos we got as input in non-registers
 *  - pseudos we've already saved off earlier..
 */
static int can_regenerate(struct bb_state *state, pseudo_t pseudo)
{
        struct storage_hash *in;

        switch (pseudo->type) {
        case PSEUDO_VAL:
        case PSEUDO_SYM:
                return 1;

        default:
                in = find_storage_hash(pseudo, state->inputs);
                if (in && in->storage->type != REG_REG)
                        return 1;
                in = find_storage_hash(pseudo, state->internal);
                if (in)
                        return 1;
        }
        return 0;
}

static void flush_one_pseudo(struct bb_state *state, struct hardreg *hardreg, ps
{
        struct storage_hash *out;
        struct storage *storage;

        if (can_regenerate(state, pseudo))
                return;

        output_comment(state, "flushing %s from %s", show_pseudo(pseudo), hardre
        out = find_storage_hash(pseudo, state->internal);
        if (!out) {
                out = find_storage_hash(pseudo, state->outputs);
                if (!out)
                        out = find_or_create_hash(pseudo, &state->internal);
        }
        storage = out->storage;
        switch (storage->type) {
        default:
                /*
                 * Aieee - the next user wants it in a register, but we
                 * need to flush it to memory in between. Which means that
                 * we need to allocate an internal one, dammit..
                 */
                out = find_or_create_hash(pseudo, &state->internal);
                storage = out->storage;
                /* Fall through */
        case REG_UDEF:
                alloc_stack(state, storage);
                /* Fall through */
        case REG_STACK:
                output_insn(state, "movl %s,%s", hardreg->name, show_memop(stora
                break;
        }
}

/* Flush a hardreg out to the storage it has.. */
static void flush_reg(struct bb_state *state, struct hardreg *reg)
{
        pseudo_t pseudo;
```

```
391         if (reg->busy)
392                 output_comment(state, "reg %s flushed while busy is %d!", reg->n
393         if (!reg->contains)
394                 return;
395         reg->dead = 0;
396         reg->used = 1;
397         FOR_EACH_PTR(reg->contains, pseudo) {
398                 if (CURRENT_TAG(pseudo) & TAG_DEAD)
399                         continue;
400                 if (!(CURRENT_TAG(pseudo) & TAG_DIRTY))
401                         continue;
402                 flush_one_pseudo(state, reg, pseudo);
403         } END_FOR_EACH_PTR(pseudo);
404         free_ptr_list(&reg->contains);
405 }

407 static struct storage_hash *find_pseudo_storage(struct bb_state *state, pseudo_t
408 {
409         struct storage_hash *src;

411         src = find_storage_hash(pseudo, state->internal);
412         if (!src) {
413                 src = find_storage_hash(pseudo, state->inputs);
414                 if (!src) {
415                         src = find_storage_hash(pseudo, state->outputs);
416                         /* Undefined? Screw it! */
417                         if (!src)
418                                 return NULL;

420                         /*
421                          * If we found output storage, it had better be local st
422                          * that we flushed to earlier..
423                          */
424                         if (src->storage->type != REG_STACK)
425                                 return NULL;
426                 }
427         }

429         /*
430          * Incoming pseudo with out any pre-set storage allocation?
431          * We can make up our own, and obviously prefer to get it
432          * in the register we already selected (if it hasn't been
433          * used yet).
434          */
435         if (src->storage->type == REG_UDEF) {
436                 if (reg && !reg->used) {
437                         src->storage->type = REG_REG;
438                         src->storage->regno = reg - hardregs;
439                         return NULL;
440                 }
441                 alloc_stack(state, src->storage);
442         }
443         return src;
444 }

446 static void mark_reg_dead(struct bb_state *state, pseudo_t pseudo, struct hardre
447 {
448         pseudo_t p;

450         FOR_EACH_PTR(reg->contains, p) {
451                 if (p != pseudo)
452                         continue;
453                 if (CURRENT_TAG(p) & TAG_DEAD)
454                         continue;
455                 output_comment(state, "marking pseudo %s in reg %s dead", show_p
456                 TAG_CURRENT(p, TAG_DEAD);
```



```
457                 reg->dead++;
458         } END_FOR_EACH_PTR(p);
459 }

461 static void add_pseudo_reg(struct bb_state *state, pseudo_t pseudo, struct hardr
462 {
463         output_comment(state, "added pseudo %s to reg %s", show_pseudo(pseudo),
464         add_ptr_list_tag(&reg->contains, pseudo, TAG_DIRTY);
465 }

467 static struct hardreg *preferred_reg(struct bb_state *state, pseudo_t target)
468 {
469         struct storage_hash *dst;

471         dst = find_storage_hash(target, state->outputs);
472         if (dst) {
473                 struct storage *storage = dst->storage;
474                 if (storage->type == REG_REG)
475                         return hardregs + storage->regno;
476         }
477         return NULL;
478 }

480 static struct hardreg *empty_reg(struct bb_state *state)
481 {
482         int i;
483         struct hardreg *reg = hardregs;

485         for (i = 0; i < REGNO; i++, reg++) {
486                 if (!reg->contains)
487                         return reg;
488         }
489         return NULL;
490 }

492 static struct hardreg *target_reg(struct bb_state *state, pseudo_t pseudo, pseud
493 {
494         int i;
495         int unable_to_find_reg = 0;
496         struct hardreg *reg;

498         /* First, see if we have a preferred target register.. */
499         reg = preferred_reg(state, target);
500         if (reg && !reg->contains)
501                 goto found;

503         reg = empty_reg(state);
504         if (reg)
505                 goto found;

507         i = last_reg;
508         do {
509                 i++;
510                 if (i >= REGNO)
511                         i = 0;
512                 reg = hardregs + i;
513                 if (!reg->busy) {
514                         flush_reg(state, reg);
515                         last_reg = i;
516                         goto found;
517                 }
518         } while (i != last_reg);
519         assert(unable_to_find_reg);

521 found:
522         add_pseudo_reg(state, pseudo, reg);
```

```
523         return reg;
524 }

526 static struct hardreg *find_in_reg(struct bb_state *state, pseudo_t pseudo)
527 {
528         int i;
529         struct hardreg *reg;

531         for (i = 0; i < REGNO; i++) {
532                 pseudo_t p;

534                 reg = hardregs + i;
535                 FOR_EACH_PTR(reg->contains, p) {
536                         if (p == pseudo) {
537                                 last_reg = i;
538                                 output_comment(state, "found pseudo %s in reg %s
539                                 return reg;
540                         }
541                 } END_FOR_EACH_PTR(p);
542         }
543         return NULL;
544 }

546 static void flush_pseudo(struct bb_state *state, pseudo_t pseudo, struct storage
547 {
548         struct hardreg *reg = find_in_reg(state, pseudo);

550         if (reg)
551                 flush_reg(state, reg);
552 }

554 static void flush_cc_cache_to_reg(struct bb_state *state, pseudo_t pseudo, struc
555 {
556         int opcode = state->cc_opcode;

558         state->cc_opcode = 0;
559         state->cc_target = NULL;
560         output_insn(state, "%s %s", opcodes[opcode], reg->name);
561 }

563 static void flush_cc_cache(struct bb_state *state)
564 {
565         pseudo_t pseudo = state->cc_target;

567         if (pseudo) {
568                 struct hardreg *dst;

570                 state->cc_target = NULL;

572                 if (!state->cc_dead) {
573                         dst = target_reg(state, pseudo, pseudo);
574                         flush_cc_cache_to_reg(state, pseudo, dst);
575                 }
576         }
577 }

579 static void add_cc_cache(struct bb_state *state, int opcode, pseudo_t pseudo)
580 {
581         assert(!state->cc_target);
582         state->cc_target = pseudo;
583         state->cc_opcode = opcode;
584         state->cc_dead = 0;
585         output_comment(state, "caching %s", opcodes[opcode]);
586 }

588 /* Fill a hardreg with the pseudo it has */
```



```
589 static struct hardreg *fill_reg(struct bb_state *state, struct hardreg *hardreg,
590 {
591         struct storage_hash *src;
592         struct instruction *def;

594         if (state->cc_target == pseudo) {
595                 flush_cc_cache_to_reg(state, pseudo, hardreg);
596                 return hardreg;
597         }

599         switch (pseudo->type) {
600         case PSEUDO_VAL:
601                 output_insn(state, "movl $%lld,%s", pseudo->value, hardreg->name
602                 break;
603         case PSEUDO_SYM:
604                 src = find_pseudo_storage(state, pseudo, NULL);
605                 /* Static thing? */
606                 if (!src) {
607                         output_insn(state, "movl $<%s>,%s", show_pseudo(pseudo),
608                         break;
609                 }
610                 switch (src->storage->type) {
611                 case REG_REG:
612                         /* Aiaiaiaiaii! Need to flush it to temporary memory */
613                         src = find_or_create_hash(pseudo, &state->internal);
614                         /* Fall through */
615                 default:
616                         alloc_stack(state, src->storage);
617                         /* Fall through */
618                 case REG_STACK:
619                 case REG_FRAME:
620                         flush_pseudo(state, pseudo, src->storage);
621                         output_insn(state, "leal %s,%s", show_memop(src->storage
622                         break;
623                 }
624                 break;
625         case PSEUDO_ARG:
626         case PSEUDO_REG:
627                 def = pseudo->def;
628                 if (def && def->opcode == OP_SETVAL) {
629                         output_insn(state, "movl $<%s>,%s", show_pseudo(def->tar
630                         break;
631                 }
632                 src = find_pseudo_storage(state, pseudo, hardreg);
633                 if (!src)
634                         break;
635                 if (src->flags & TAG_DEAD)
636                         mark_reg_dead(state, pseudo, hardreg);
637                 output_insn(state, "mov.%d %s,%s", 32, show_memop(src->storage),
638                 break;
639         default:
640                 output_insn(state, "reload %s from %s", hardreg->name, show_pseu
641                 break;
642         }
643         return hardreg;
644 }

646 static struct hardreg *getreg(struct bb_state *state, pseudo_t pseudo, pseudo_t
647 {
648         struct hardreg *reg;

650         reg = find_in_reg(state, pseudo);
651         if (reg)
652                 return reg;
653         reg = target_reg(state, pseudo, target);
654         return fill_reg(state, reg, pseudo);
```

```
655 }

657 static void move_reg(struct bb_state *state, struct hardreg *src, struct hardreg
658 {
659         output_insn(state, "movl %s,%s", src->name, dst->name);
660 }

662 static struct hardreg *copy_reg(struct bb_state *state, struct hardreg *src, pse
663 {
664         int i;
665         struct hardreg *reg;

667         /* If the container has been killed off, just re-use it */
668         if (!src->contains)
669                 return src;

671         /* If "src" only has one user, and the contents are dead, we can re-use
672         if (src->busy == 1 && src->dead == 1)
673                 return src;

675         reg = preferred_reg(state, target);
676         if (reg && !reg->contains) {
677                 output_comment(state, "copying %s to preferred target %s", show_
678                 move_reg(state, src, reg);
679                 return reg;
680         }

682         for (i = 0; i < REGNO; i++) {
683                 reg = hardregs + i;
684                 if (!reg->contains) {
685                         output_comment(state, "copying %s to %s", show_pseudo(ta
686                         output_insn(state, "movl %s,%s", src->name, reg->name);
687                         return reg;
688                 }
689         }

691         flush_reg(state, src);
692         return src;
693 }

695 static void put_operand(struct bb_state *state, struct operand *op)
696 {
697         switch (op->type) {
698         case OP_REG:
699                 op->reg->busy--;
700                 break;
701         case OP_ADDR:
702         case OP_MEM:
703                 if (op->base)
704                         op->base->busy--;
705                 if (op->index)
706                         op->index->busy--;
707                 break;
708         default:
709                 break;
710         }
711 }

713 static struct operand *alloc_op(void)
714 {
715         struct operand *op = malloc(sizeof(*op));
716         memset(op, 0, sizeof(*op));
717         return op;
718 }

720 static struct operand *get_register_operand(struct bb_state *state, pseudo_t pse
```



```
721 {
722         struct operand *op = alloc_op();
723         op->type = OP_REG;
724         op->reg = getreg(state, pseudo, target);
725         op->reg->busy++;
726         return op;
727 }

729 static int get_sym_frame_offset(struct bb_state *state, pseudo_t pseudo)
730 {
731         int offset = pseudo->nr;
732         if (offset < 0) {
733                 offset = alloc_stack_offset(4);
734                 pseudo->nr = offset;
735         }
736         return offset;
737 }

739 static struct operand *get_generic_operand(struct bb_state *state, pseudo_t pseu
740 {
741         struct hardreg *reg;
742         struct storage *src;
743         struct storage_hash *hash;
744         struct operand *op = malloc(sizeof(*op));

746         memset(op, 0, sizeof(*op));
747         switch (pseudo->type) {
748         case PSEUDO_VAL:
749                 op->type = OP_VAL;
750                 op->value = pseudo->value;
751                 break;

753         case PSEUDO_SYM: {
754                 struct symbol *sym = pseudo->sym;
755                 op->type = OP_ADDR;
756                 if (sym->ctype.modifiers & MOD_NONLOCAL) {
757                         op->sym = sym;
758                         break;
759                 }
760                 op->base = hardregs + REG_EBP;
761                 op->offset = get_sym_frame_offset(state, pseudo);
762                 break;
763         }

765         default:
766                 reg = find_in_reg(state, pseudo);
767                 if (reg) {
768                         op->type = OP_REG;
769                         op->reg = reg;
770                         reg->busy++;
771                         break;
772                 }
773                 hash = find_pseudo_storage(state, pseudo, NULL);
774                 if (!hash)
775                         break;
776                 src = hash->storage;
777                 switch (src->type) {
778                 case REG_REG:
779                         op->type = OP_REG;
780                         op->reg = hardregs + src->regno;
781                         op->reg->busy++;
782                         break;
783                 case REG_FRAME:
784                         op->type = OP_MEM;
785                         op->offset = src->offset;
786                         op->base = hardregs + REG_EBP;
```

```
787                         break;
788                 case REG_STACK:
789                         op->type = OP_MEM;
790                         op->offset = src->offset;
791                         op->base = hardregs + REG_ESP;
792                         break;
793                 default:
794                         break;
795                 }
796         }
797         return op;
798 }

800 /* Callers should be made to use the proper "operand" formats */
801 static const char *generic(struct bb_state *state, pseudo_t pseudo)
802 {
803         struct hardreg *reg;
804         struct operand *op = get_generic_operand(state, pseudo);
805         static char buf[100];
806         const char *str;

808         switch (op->type) {
809         case OP_ADDR:
810                 if (!op->offset && op->base && !op->sym)
811                         return op->base->name;
812                 if (op->sym && !op->base) {
813                         int len = sprintf(buf, "$ %s", show_op(state, op));
814                         if (op->offset)
815                                 sprintf(buf + len, " + %d", op->offset);
816                         return buf;
817                 }
818                 str = show_op(state, op);
819                 put_operand(state, op);
820                 reg = target_reg(state, pseudo, NULL);
821                 output_insn(state, "lea %s,%s", show_op(state, op), reg->name);
822                 return reg->name;

824         default:
825                 str = show_op(state, op);
826         }
827         put_operand(state, op);
828         return str;
829 }

831 static struct operand *get_address_operand(struct bb_state *state, struct instru
832 {
833         struct hardreg *base;
834         struct operand *op = get_generic_operand(state, memop->src);

836         switch (op->type) {
837         case OP_ADDR:
838                 op->offset += memop->offset;
839                 break;
840         default:
841                 put_operand(state, op);
842                 base = getreg(state, memop->src, NULL);
843                 op->type = OP_ADDR;
844                 op->base = base;
845                 base->busy++;
846                 op->offset = memop->offset;
847                 op->sym = NULL;
848         }
849         return op;
850 }

852 static const char *address(struct bb_state *state, struct instruction *memop)
```



```
853 {
854         struct operand *op = get_address_operand(state, memop);
855         const char *str = show_op(state, op);
856         put_operand(state, op);
857         return str;
858 }

860 static const char *reg_or_imm(struct bb_state *state, pseudo_t pseudo)
861 {
862         switch(pseudo->type) {
863         case PSEUDO_VAL:
864                 return show_pseudo(pseudo);
865         default:
866                 return getreg(state, pseudo, NULL)->name;
867         }
868 }

870 static void kill_dead_reg(struct hardreg *reg)
871 {
872         if (reg->dead) {
873                 pseudo_t p;
874
875                 FOR_EACH_PTR(reg->contains, p) {
876                         if (CURRENT_TAG(p) & TAG_DEAD) {
877                                 DELETE_CURRENT_PTR(p);
878                                 reg->dead--;
879                         }
880                 } END_FOR_EACH_PTR(p);
881                 PACK_PTR_LIST(&reg->contains);
882                 assert(!reg->dead);
883         }
884 }

886 static struct hardreg *target_copy_reg(struct bb_state *state, struct hardreg *s
887 {
888         kill_dead_reg(src);
889         return copy_reg(state, src, target);
890 }

892 static void do_binop(struct bb_state *state, struct instruction *insn, pseudo_t
893 {
894         const char *op = opcodes[insn->opcode];
895         struct operand *src = get_register_operand(state, val1, insn->target);
896         struct operand *src2 = get_generic_operand(state, val2);
897         struct hardreg *dst;

899         dst = target_copy_reg(state, src->reg, insn->target);
900         output_insn(state, "%s.%d %s,%s", op, insn->size, show_op(state, src2),
901         put_operand(state, src);
902         put_operand(state, src2);
903         add_pseudo_reg(state, insn->target, dst);
904 }

906 static void generate_binop(struct bb_state *state, struct instruction *insn)
907 {
908         flush_cc_cache(state);
909         do_binop(state, insn, insn->src1, insn->src2);
910 }

912 static int is_dead_reg(struct bb_state *state, pseudo_t pseudo, struct hardreg *
913 {
914         pseudo_t p;
915         FOR_EACH_PTR(reg->contains, p) {
916                 if (p == pseudo)
917                         return CURRENT_TAG(p) & TAG_DEAD;
918         } END_FOR_EACH_PTR(p);
```

```
919         return 0;
920 }

922 /*
923  * Commutative binops are much more flexible, since we can switch the
924  * sources around to satisfy the target register, or to avoid having
925  * to load one of them into a register..
926  */
927 static void generate_commutative_binop(struct bb_state *state, struct instructio
928 {
929         pseudo_t src1, src2;
930         struct hardreg *reg1, *reg2;

932         flush_cc_cache(state);
933         src1 = insn->src1;
934         src2 = insn->src2;
935         reg2 = find_in_reg(state, src2);
936         if (!reg2)
937                 goto dont_switch;
938         reg1 = find_in_reg(state, src1);
939         if (!reg1)
940                 goto do_switch;
941         if (!is_dead_reg(state, src2, reg2))
942                 goto dont_switch;
943         if (!is_dead_reg(state, src1, reg1))
944                 goto do_switch;

946         /* Both are dead. Is one preferable? */
947         if (reg2 != preferred_reg(state, insn->target))
948                 goto dont_switch;

950 do_switch:
951         src1 = src2;
952         src2 = insn->src1;
953 dont_switch:
954         do_binop(state, insn, src1, src2);
955 }

957 /*
958  * This marks a pseudo dead. It still stays on the hardreg list (the hardreg
959  * still has its value), but it's scheduled to be killed after the next
960  * "sequence point" when we call "kill_read_pseudos()"
961  */
962 static void mark_pseudo_dead(struct bb_state *state, pseudo_t pseudo)
963 {
964         int i;
965         struct storage_hash *src;

967         if (state->cc_target == pseudo)
968                 state->cc_dead = 1;
969         src = find_pseudo_storage(state, pseudo, NULL);
970         if (src)
971                 src->flags |= TAG_DEAD;
972         for (i = 0; i < REGNO; i++)
973                 mark_reg_dead(state, pseudo, hardregs + i);
974 }

976 static void kill_dead_pseudos(struct bb_state *state)
977 {
978         int i;

980         for (i = 0; i < REGNO; i++) {
981                 kill_dead_reg(hardregs + i);
982         }
983 }
```



```
985 static void generate_store(struct instruction *insn, struct bb_state *state)
986 {
987         output_insn(state, "mov.%d %s,%s", insn->size, reg_or_imm(state, insn->t
988 }

990 static void generate_load(struct instruction *insn, struct bb_state *state)
991 {
992         const char *input = address(state, insn);
993         struct hardreg *dst;

995         kill_dead_pseudos(state);
996         dst = target_reg(state, insn->target, NULL);
997         output_insn(state, "mov.%d %s,%s", insn->size, input, dst->name);
998 }

1000 static void kill_pseudo(struct bb_state *state, pseudo_t pseudo)
1001 {
1002         int i;
1003         struct hardreg *reg;

1005         output_comment(state, "killing pseudo %s", show_pseudo(pseudo));
1006         for (i = 0; i < REGNO; i++) {
1007                 pseudo_t p;

1009                 reg = hardregs + i;
1010                 FOR_EACH_PTR(reg->contains, p) {
1011                         if (p != pseudo)
1012                                 continue;
1013                         if (CURRENT_TAG(p) & TAG_DEAD)
1014                                 reg->dead--;
1015                         output_comment(state, "removing pseudo %s from reg %s",
1016                                 show_pseudo(pseudo), reg->name);
1017                         DELETE_CURRENT_PTR(p);
1018                 } END_FOR_EACH_PTR(p);
1019                 PACK_PTR_LIST(&reg->contains);
1020         }
1021 }

1023 static void generate_copy(struct bb_state *state, struct instruction *insn)
1024 {
1025         struct hardreg *src = getreg(state, insn->src, insn->target);
1026         kill_pseudo(state, insn->target);
1027         add_pseudo_reg(state, insn->target, src);
1028 }

1030 static void generate_cast(struct bb_state *state, struct instruction *insn)
1031 {
1032         struct hardreg *src = getreg(state, insn->src, insn->target);
1033         struct hardreg *dst;
1034         unsigned int old = insn->orig_type ? insn->orig_type->bit_size : 0;
1035         unsigned int new = insn->size;

1037         /*
1038          * Cast to smaller type? Ignore the high bits, we
1039          * just keep both pseudos in the same register.
1040          */
1041         if (old >= new) {
1042                 add_pseudo_reg(state, insn->target, src);
1043                 return;
1044         }

1046         dst = target_copy_reg(state, src, insn->target);

1048         if (insn->orig_type && (insn->orig_type->ctype.modifiers & MOD_SIGNED))
1049                 output_insn(state, "sext.%d.%d %s", old, new, dst->name);
1050         } else {
```

```
1051                 unsigned long long mask;
1052                 mask = ~(~0ULL << old);
1053                 mask &= ~(~0ULL << new);
1054                 output_insn(state, "andl.%d $%#llx,%s", insn->size, mask, dst->n
1055         }
1056         add_pseudo_reg(state, insn->target, dst);
1057 }

1059 static void generate_output_storage(struct bb_state *state);

1061 static const char *conditional[] = {
1062         [OP_SET_EQ] = "e",
1063         [OP_SET_NE] = "ne",
1064         [OP_SET_LE] = "le",
1065         [OP_SET_GE] = "ge",
1066         [OP_SET_LT] = "lt",
1067         [OP_SET_GT] = "gt",
1068         [OP_SET_B] = "b",
1069         [OP_SET_A] = "a",
1070         [OP_SET_BE] = "be",
1071         [OP_SET_AE] = "ae"
1072 };
1073

1075 static void generate_branch(struct bb_state *state, struct instruction *br)
1076 {
1077         const char *cond = "XXX";
1078         struct basic_block *target;

1080         if (br->cond) {
1081                 if (state->cc_target == br->cond) {
1082                         cond = conditional[state->cc_opcode];
1083                 } else {
1084                         struct hardreg *reg = getreg(state, br->cond, NULL);
1085                         output_insn(state, "testl %s,%s", reg->name, reg->name);
1086                         cond = "ne";
1087                 }
1088         }
1089         generate_output_storage(state);
1090         target = br->bb_true;
1091         if (br->cond) {
1092                 output_insn(state, "j%s .L%p", cond, target);
1093                 target = br->bb_false;
1094         }
1095         output_insn(state, "jmp .L%p", target);
1096 }

1098 /* We've made sure that there is a dummy reg live for the output */
1099 static void generate_switch(struct bb_state *state, struct instruction *insn)
1100 {
1101         struct hardreg *reg = hardregs + SWITCH_REG;

1103         generate_output_storage(state);
1104         output_insn(state, "switch on %s", reg->name);
1105         output_insn(state, "unimplemented: %s", show_instruction(insn));
1106 }

1108 static void generate_ret(struct bb_state *state, struct instruction *ret)
1109 {
1110         if (ret->src && ret->src != VOID) {
1111                 struct hardreg *wants = hardregs+0;
1112                 struct hardreg *reg = getreg(state, ret->src, NULL);
1113                 if (reg != wants)
1114                         output_insn(state, "movl %s,%s", reg->name, wants->name)
1115         }
1116         output_insn(state, "ret");
```



```
1117 }

1119 /*
1120  * Fake "call" linearization just as a taster..
1121  */
1122 static void generate_call(struct bb_state *state, struct instruction *insn)
1123 {
1124         int offset = 0;
1125         pseudo_t arg;

1127         FOR_EACH_PTR(insn->arguments, arg) {
1128                 output_insn(state, "pushl %s", generic(state, arg));
1129                 offset += 4;
1130         } END_FOR_EACH_PTR(arg);
1131         flush_reg(state, hardregs+0);
1132         flush_reg(state, hardregs+1);
1133         flush_reg(state, hardregs+2);
1134         output_insn(state, "call %s", show_pseudo(insn->func));
1135         if (offset)
1136                 output_insn(state, "addl $%d,%%esp", offset);
1137         if (insn->target && insn->target != VOID)
1138                 add_pseudo_reg(state, insn->target, hardregs+0);
1139 }

1141 static void generate_select(struct bb_state *state, struct instruction *insn)
1142 {
1143         const char *cond;
1144         struct hardreg *src1, *src2, *dst;

1146         src1 = getreg(state, insn->src2, NULL);
1147         dst = copy_reg(state, src1, insn->target);
1148         add_pseudo_reg(state, insn->target, dst);
1149         src2 = getreg(state, insn->src3, insn->target);

1151         if (state->cc_target == insn->src1) {
1152                 cond = conditional[state->cc_opcode];
1153         } else {
1154                 struct hardreg *reg = getreg(state, insn->src1, NULL);
1155                 output_insn(state, "testl %s,%s", reg->name, reg->name);
1156                 cond = "ne";
1157         }

1159         output_insn(state, "sel%s %s,%s", cond, src2->name, dst->name);
1160 }

1162 struct asm_arg {
1163         const struct ident *name;
1164         const char *value;
1165         pseudo_t pseudo;
1166         struct hardreg *reg;
1167 };

1169 static void replace_asm_arg(char **dst_p, struct asm_arg *arg)
1170 {
1171         char *dst = *dst_p;
1172         int len = strlen(arg->value);

1174         memcpy(dst, arg->value, len);
1175         *dst_p = dst + len;
1176 }

1178 static void replace_asm_percent(const char **src_p, char **dst_p, struct asm_arg
1179 {
1180         const char *src = *src_p;
1181         char c;
1182         int index;
```

```
1184         c = *src++;
1185         switch (c) {
1186         case '0' ... '9':
1187                 index = c - '0';
1188                 if (index < nr)
1189                         replace_asm_arg(dst_p, args+index);
1190                 break;
1191         }
1192         *src_p = src;
1193         return;
1194 }

1196 static void replace_asm_named(const char **src_p, char **dst_p, struct asm_arg *
1197 {
1198         const char *src = *src_p;
1199         const char *end = src;

1201         for(;;) {
1202                 char c = *end++;
1203                 if (!c)
1204                         return;
1205                 if (c == ']') {
1206                         int i;

1208                         *src_p = end;
1209                         for (i = 0; i < nr; i++) {
1210                                 const struct ident *ident = args[i].name;
1211                                 int len;
1212                                 if (!ident)
1213                                         continue;
1214                                 len = ident->len;
1215                                 if (memcmp(src, ident->name, len))
1216                                         continue;
1217                                 replace_asm_arg(dst_p, args+i);
1218                                 return;
1219                         }
1220                 }
1221         }
1222 }

1224 static const char *replace_asm_args(const char *str, struct asm_arg *args, int n
1225 {
1226         static char buffer[1000];
1227         char *p = buffer;

1229         for (;;) {
1230                 char c = *str;
1231                 *p = c;
1232                 if (!c)
1233                         return buffer;
1234                 str++;
1235                 switch (c) {
1236                 case '%':
1237                         if (*str == '%') {
1238                                 str++;
1239                                 p++;
1240                                 continue;
1241                         }
1242                         replace_asm_percent(&str, &p, args, nr);
1243                         continue;
1244                 case '[':
1245                         replace_asm_named(&str, &p, args, nr);
1246                         continue;
1247                 default:
1248                         break;
```



```
1249                 }
1250                 p++;
1251         }
1252 }

1254 #define MAX_ASM_ARG (50)
1255 static struct asm_arg asm_arguments[MAX_ASM_ARG];

1257 static struct asm_arg *generate_asm_inputs(struct bb_state *state, struct asm_co
1258 {
1259         struct asm_constraint *entry;

1261         FOR_EACH_PTR(list, entry) {
1262                 const char *constraint = entry->constraint;
1263                 pseudo_t pseudo = entry->pseudo;
1264                 struct hardreg *reg, *orig;
1265                 const char *string;
1266                 int index;

1268                 string = "undef";
1269                 switch (*constraint) {
1270                 case 'r':
1271                         string = getreg(state, pseudo, NULL)->name;
1272                         break;
1273                 case '0' ... '9':
1274                         index = *constraint - '0';
1275                         reg = asm_arguments[index].reg;
1276                         orig = find_in_reg(state, pseudo);
1277                         if (orig)
1278                                 move_reg(state, orig, reg);
1279                         else
1280                                 fill_reg(state, reg, pseudo);
1281                         string = reg->name;
1282                         break;
1283                 default:
1284                         string = generic(state, pseudo);
1285                         break;
1286                 }

1288                 output_insn(state, "# asm input \"%s\": %s : %s", constraint, sh

1290                 arg->name = entry->ident;
1291                 arg->value = string;
1292                 arg->pseudo = NULL;
1293                 arg->reg = NULL;
1294                 arg++;
1295         } END_FOR_EACH_PTR(entry);
1296         return arg;
1297 }

1299 static struct asm_arg *generate_asm_outputs(struct bb_state *state, struct asm_c
1300 {
1301         struct asm_constraint *entry;

1303         FOR_EACH_PTR(list, entry) {
1304                 const char *constraint = entry->constraint;
1305                 pseudo_t pseudo = entry->pseudo;
1306                 struct hardreg *reg;
1307                 const char *string;

1309                 while (*constraint == '=' || *constraint == '+')
1310                         constraint++;

1312                 string = "undef";
1313                 switch (*constraint) {
1314                 case 'r':
```

```
1315                 default:
1316                         reg = target_reg(state, pseudo, NULL);
1317                         arg->pseudo = pseudo;
1318                         arg->reg = reg;
1319                         string = reg->name;
1320                         break;
1321                 }

1323                 output_insn(state, "# asm output \"%s\": %s : %s", constraint, s

1325                 arg->name = entry->ident;
1326                 arg->value = string;
1327                 arg++;
1328         } END_FOR_EACH_PTR(entry);
1329         return arg;
1330 }

1332 static void generate_asm(struct bb_state *state, struct instruction *insn)
1333 {
1334         const char *str = insn->string;

1336         if (insn->asm_rules->outputs || insn->asm_rules->inputs) {
1337                 struct asm_arg *arg;

1339                 arg = generate_asm_outputs(state, insn->asm_rules->outputs, asm_
1340                 arg = generate_asm_inputs(state, insn->asm_rules->inputs, arg);
1341                 str = replace_asm_args(str, asm_arguments, arg - asm_arguments);
1342         }
1343         output_insn(state, "%s", str);
1344 }

1346 static void generate_compare(struct bb_state *state, struct instruction *insn)
1347 {
1348         struct hardreg *src;
1349         const char *src2;
1350         int opcode;

1352         flush_cc_cache(state);
1353         opcode = insn->opcode;

1355         /*
1356          * We should try to switch these around if necessary,
1357          * and update the opcode to match..
1358          */
1359         src = getreg(state, insn->src1, insn->target);
1360         src2 = generic(state, insn->src2);

1362         output_insn(state, "cmp.%d %s,%s", insn->size, src2, src->name);

1364         add_cc_cache(state, opcode, insn->target);
1365 }

1367 static void generate_one_insn(struct instruction *insn, struct bb_state *state)
1368 {
1369         if (verbose)
1370                 output_comment(state, "%s", show_instruction(insn));

1372         switch (insn->opcode) {
1373         case OP_ENTRY: {
1374                 struct symbol *sym = insn->bb->ep->name;
1375                 const char *name = show_ident(sym->ident);
1376                 if (sym->ctype.modifiers & MOD_STATIC)
1377                         printf("\n\n%s:\n", name);
1378                 else
1379                         printf("\n\n.globl %s\n%s:\n", name, name);
1380                 break;
```



```
1381         }

1383         /*
1384          * OP_SETVAL likewise doesn't actually generate any
1385          * code. On use, the "def" of the pseudo will be
1386          * looked up.
1387          */
1388         case OP_SETVAL:
1389                 break;

1391         case OP_STORE:
1392                 generate_store(insn, state);
1393                 break;

1395         case OP_LOAD:
1396                 generate_load(insn, state);
1397                 break;

1399         case OP_DEATHNOTE:
1400                 mark_pseudo_dead(state, insn->target);
1401                 return;

1403         case OP_COPY:
1404                 generate_copy(state, insn);
1405                 break;

1407         case OP_ADD: case OP_MULU: case OP_MULS:
1408         case OP_AND: case OP_OR: case OP_XOR:
1409         case OP_AND_BOOL: case OP_OR_BOOL:
1410                 generate_commutative_binop(state, insn);
1411                 break;

1413         case OP_SUB: case OP_DIVU: case OP_DIVS:
1414         case OP_MODU: case OP_MODS:
1415         case OP_SHL: case OP_LSR: case OP_ASR:
1416                 generate_binop(state, insn);
1417                 break;

1419         case OP_BINCMP ... OP_BINCMP_END:
1420                 generate_compare(state, insn);
1421                 break;

1423         case OP_CAST: case OP_SCAST: case OP_FPCAST: case OP_PTRCAST:
1424                 generate_cast(state, insn);
1425                 break;

1427         case OP_SEL:
1428                 generate_select(state, insn);
1429                 break;

1431         case OP_BR:
1432         case OP_CBR:
1433                 generate_branch(state, insn);
1434                 break;

1436         case OP_SWITCH:
1437                 generate_switch(state, insn);
1438                 break;

1440         case OP_CALL:
1441                 generate_call(state, insn);
1442                 break;

1444         case OP_RET:
1445                 generate_ret(state, insn);
1446                 break;
```

```
1448         case OP_ASM:
1449                 generate_asm(state, insn);
1450                 break;

1452         case OP_PHI:
1453         case OP_PHISOURCE:
1454         default:
1455                 output_insn(state, "unimplemented: %s", show_instruction(insn));
1456                 break;
1457         }
1458         kill_dead_pseudos(state);
1459 }

1461 #define VERY_BUSY 1000
1462 #define REG_FIXED 2000

1464 static void write_reg_to_storage(struct bb_state *state, struct hardreg *reg, ps
1465 {
1466         int i;
1467         struct hardreg *out;

1469         switch (storage->type) {
1470         case REG_REG:
1471                 out = hardregs + storage->regno;
1472                 if (reg == out)
1473                         return;
1474                 output_insn(state, "movl %s,%s", reg->name, out->name);
1475                 return;
1476         case REG_UDEF:
1477                 if (reg->busy < VERY_BUSY) {
1478                         storage->type = REG_REG;
1479                         storage->regno = reg - hardregs;
1480                         reg->busy = REG_FIXED;
1481                         return;
1482                 }

1484                 /* Try to find a non-busy register.. */
1485                 for (i = 0; i < REGNO; i++) {
1486                         out = hardregs + i;
1487                         if (out->contains)
1488                                 continue;
1489                         output_insn(state, "movl %s,%s", reg->name, out->name);
1490                         storage->type = REG_REG;
1491                         storage->regno = i;
1492                         out->busy = REG_FIXED;
1493                         return;
1494                 }

1496                 /* Fall back on stack allocation ... */
1497                 alloc_stack(state, storage);
1498                 /* Fall through */
1499         default:
1500                 output_insn(state, "movl %s,%s", reg->name, show_memop(storage))
1501                 return;
1502         }
1503 }

1505 static void write_val_to_storage(struct bb_state *state, pseudo_t src, struct st
1506 {
1507         struct hardreg *out;

1509         switch (storage->type) {
1510         case REG_UDEF:
1511                 alloc_stack(state, storage);
1512         default:
```



```
1513                 output_insn(state, "movl %s,%s", show_pseudo(src), show_memop(st
1514                 break;
1515         case REG_REG:
1516                 out = hardregs + storage->regno;
1517                 output_insn(state, "movl %s,%s", show_pseudo(src), out->name);
1518         }
1519 }

1521 static void fill_output(struct bb_state *state, pseudo_t pseudo, struct storage
1522 {
1523         int i;
1524         struct storage_hash *in;
1525         struct instruction *def;

1527         /* Is that pseudo a constant value? */
1528         switch (pseudo->type) {
1529         case PSEUDO_VAL:
1530                 write_val_to_storage(state, pseudo, out);
1531                 return;
1532         case PSEUDO_REG:
1533                 def = pseudo->def;
1534                 if (def && def->opcode == OP_SETVAL) {
1535                         write_val_to_storage(state, pseudo, out);
1536                         return;
1537                 }
1538         default:
1539                 break;
1540         }

1542         /* See if we have that pseudo in a register.. */
1543         for (i = 0; i < REGNO; i++) {
1544                 struct hardreg *reg = hardregs + i;
1545                 pseudo_t p;

1547                 FOR_EACH_PTR(reg->contains, p) {
1548                         if (p == pseudo) {
1549                                 write_reg_to_storage(state, reg, pseudo, out);
1550                                 return;
1551                         }
1552                 } END_FOR_EACH_PTR(p);
1553         }

1555         /* Do we have it in another storage? */
1556         in = find_storage_hash(pseudo, state->internal);
1557         if (!in) {
1558                 in = find_storage_hash(pseudo, state->inputs);
1559                 /* Undefined? */
1560                 if (!in)
1561                         return;
1562         }
1563         switch (out->type) {
1564         case REG_UDEF:
1565                 *out = *in->storage;
1566                 break;
1567         case REG_REG:
1568                 output_insn(state, "movl %s,%s", show_memop(in->storage), hardre
1569                 break;
1570         default:
1571                 if (out == in->storage)
1572                         break;
1573                 if ((out->type == in->storage->type) && (out->regno == in->stora
1574                         break;
1575                 output_insn(state, "movl %s,%s", show_memop(in->storage), show_m
1576                 break;
1577         }
1578         return;
```

```
}

static int final_pseudo_flush(struct bb_state *state, pseudo_t pseudo, struct ha
{
        struct storage_hash *hash;
        struct storage *out;
        struct hardreg *dst;

        /*
         * Since this pseudo is live at exit, we'd better have output
         * storage for it..
         */
        hash = find_storage_hash(pseudo, state->outputs);
        if (!hash)
                return 1;
        out = hash->storage;

        /* If the output is in a register, try to get it there.. */
        if (out->type == REG_REG) {
                dst = hardregs + out->regno;
                /*
                 * Two good cases: nobody is using the right register,
                 * or we've already set it aside for output..
                 */
                if (!dst->contains || dst->busy > VERY_BUSY)
                        goto copy_to_dst;

                /* Aiee. Try to keep it in a register.. */
                dst = empty_reg(state);
                if (dst)
                        goto copy_to_dst;

                return 0;
        }

        /* If the output is undefined, let's see if we can put it in a register.
        if (out->type == REG_UDEF) {
                dst = empty_reg(state);
                if (dst) {
                        out->type = REG_REG;
                        out->regno = dst - hardregs;
                        goto copy_to_dst;
                }
                /* Uhhuh. Not so good. No empty registers right now */
                return 0;
        }

        /* If we know we need to flush it, just do so already .. */
        output_insn(state, "movl %s,%s", reg->name, show_memop(out));
        return 1;

copy_to_dst:
        if (reg == dst)
                return 1;
        output_insn(state, "movl %s,%s", reg->name, dst->name);
        add_pseudo_reg(state, pseudo, dst);
        return 1;
}

/*
 * This tries to make sure that we put all the pseudos that are
 * live on exit into the proper storage
 */
static void generate_output_storage(struct bb_state *state)
{
        struct storage_hash *entry;
```



```
        /* Go through the fixed outputs, making sure we have those regs free */
        FOR_EACH_PTR(state->outputs, entry) {
                struct storage *out = entry->storage;
                if (out->type == REG_REG) {
                        struct hardreg *reg = hardregs + out->regno;
                        pseudo_t p;
                        int flushme = 0;

                        reg->busy = REG_FIXED;
                        FOR_EACH_PTR(reg->contains, p) {
                                if (p == entry->pseudo) {
                                        flushme = -100;
                                        continue;
                                }
                                if (CURRENT_TAG(p) & TAG_DEAD)
                                        continue;

                                /* Try to write back the pseudo to where it shou
                                if (final_pseudo_flush(state, p, reg)) {
                                        DELETE_CURRENT_PTR(p);
                                        continue;
                                }
                                flushme++;
                        } END_FOR_EACH_PTR(p);
                        PACK_PTR_LIST(&reg->contains);
                        if (flushme > 0)
                                flush_reg(state, reg);
                }
        } END_FOR_EACH_PTR(entry);

        FOR_EACH_PTR(state->outputs, entry) {
                fill_output(state, entry->pseudo, entry->storage);
        } END_FOR_EACH_PTR(entry);
}

static void generate(struct basic_block *bb, struct bb_state *state)
{
        int i;
        struct storage_hash *entry;
        struct instruction *insn;

        for (i = 0; i < REGNO; i++) {
                free_ptr_list(&hardregs[i].contains);
                hardregs[i].busy = 0;
                hardregs[i].dead = 0;
                hardregs[i].used = 0;
        }

        FOR_EACH_PTR(state->inputs, entry) {
                struct storage *storage = entry->storage;
                const char *name = show_storage(storage);
                output_comment(state, "incoming %s in %s", show_pseudo(entry->ps
                if (storage->type == REG_REG) {
                        int regno = storage->regno;
                        add_pseudo_reg(state, entry->pseudo, hardregs + regno);
                        name = hardregs[regno].name;
                }
        } END_FOR_EACH_PTR(entry);

        output_label(state, ".L%p", bb);
        FOR_EACH_PTR(bb->insns, insn) {
                if (!insn->bb)
                        continue;
                generate_one_insn(insn, state);
        } END_FOR_EACH_PTR(insn);
```

```
1712         if (verbose) {
1713                 output_comment(state, "--- in ---");
1714                 FOR_EACH_PTR(state->inputs, entry) {
1715                         output_comment(state, "%s <- %s", show_pseudo(entry->pse
1716                 } END_FOR_EACH_PTR(entry);
1717                 output_comment(state, "--- spill ---");
1718                 FOR_EACH_PTR(state->internal, entry) {
1719                         output_comment(state, "%s <-> %s", show_pseudo(entry->ps
1720                 } END_FOR_EACH_PTR(entry);
1721                 output_comment(state, "--- out ---");
1722                 FOR_EACH_PTR(state->outputs, entry) {
1723                         output_comment(state, "%s -> %s", show_pseudo(entry->pse
1724                 } END_FOR_EACH_PTR(entry);
1725         }
1726         printf("\n");
1727 }

1729 static void generate_list(struct basic_block_list *list, unsigned long generatio
1730 {
1731         struct basic_block *bb;
1732         FOR_EACH_PTR(list, bb) {
1733                 if (bb->generation == generation)
1734                         continue;
1735                 output_bb(bb, generation);
1736         } END_FOR_EACH_PTR(bb);
1737 }

1739 /*
1740  * Mark all the output registers of all the parents
1741  * as being "used" - this does not mean that we cannot
1742  * re-use them, but it means that we cannot ask the
1743  * parents to pass in another pseudo in one of those
1744  * registers that it already uses for another child.
1745  */
1746 static void mark_used_registers(struct basic_block *bb, struct bb_state *state)
1747 {
1748         struct basic_block *parent;

1750         FOR_EACH_PTR(bb->parents, parent) {
1751                 struct storage_hash_list *outputs = gather_storage(parent, STOR_
1752                 struct storage_hash *entry;

1754                 FOR_EACH_PTR(outputs, entry) {
1755                         struct storage *s = entry->storage;
1756                         if (s->type == REG_REG) {
1757                                 struct hardreg *reg = hardregs + s->regno;
1758                                 reg->used = 1;
1759                         }
1760                 } END_FOR_EACH_PTR(entry);
1761         } END_FOR_EACH_PTR(parent);
1762 }

1764 static void output_bb(struct basic_block *bb, unsigned long generation)
1765 {
1766         struct bb_state state;

1768         bb->generation = generation;

1770         /* Make sure all parents have been generated first */
1771         generate_list(bb->parents, generation);

1773         state.pos = bb->pos;
1774         state.inputs = gather_storage(bb, STOR_IN);
1775         state.outputs = gather_storage(bb, STOR_OUT);
1776         state.internal = NULL;
```



```
1777         state.cc_opcode = 0;
1778         state.cc_target = NULL;

1780         /* Mark incoming registers used */
1781         mark_used_registers(bb, &state);

1783         generate(bb, &state);

1785         free_ptr_list(&state.inputs);
1786         free_ptr_list(&state.outputs);

1788         /* Generate all children... */
1789         generate_list(bb->children, generation);
1790 }

1792 /*
1793  * We should set up argument sources here..
1794  *
1795  * Things like "first three arguments in registers" etc
1796  * are all for this place.
1797  *
1798  * On x86, we default to stack, unless it's a static
1799  * function that doesn't have its address taken.
1800  *
1801  * I should implement the -mregparm=X cmd line option.
1802  */
1803 static void set_up_arch_entry(struct entrypoint *ep, struct instruction *entry)
1804 {
1805         pseudo_t arg;
1806         struct symbol *sym, *argtype;
1807         int i, offset, regparm;

1809         sym = ep->name;
1810         regparm = 0;
1811         if (!(sym->ctype.modifiers & MOD_ADDRESSABLE))
1812                 regparm = 3;
1813         sym = sym->ctype.base_type;
1814         i = 0;
1815         offset = 0;
1816         PREPARE_PTR_LIST(sym->arguments, argtype);
1817         FOR_EACH_PTR(entry->arg_list, arg) {
1818                 struct storage *in = lookup_storage(entry->bb, arg, STOR_IN);
1819                 if (!in) {
1820                         in = alloc_storage();
1821                         add_storage(in, entry->bb, arg, STOR_IN);
1822                 }
1823                 if (i < regparm) {
1824                         in->type = REG_REG;
1825                         in->regno = i;
1826                 } else {
1827                         int bits = argtype ? argtype->bit_size : 0;

1829                         if (bits < bits_in_int)
1830                                 bits = bits_in_int;

1832                         in->type = REG_FRAME;
1833                         in->offset = offset;
1834
1835                         offset += bits_to_bytes(bits);
1836                 }
1837                 i++;
1838                 NEXT_PTR_LIST(argtype);
1839         } END_FOR_EACH_PTR(arg);
1840         FINISH_PTR_LIST(argtype);
1841 }
```

```
1843 /*
1844  * Set up storage information for "return"
1845  *
1846  * Not strictly necessary, since the code generator will
1847  * certainly move the return value to the right register,
1848  * but it can help register allocation if the allocator
1849  * sees that the target register is going to return in %eax.
1850  */
1851 static void set_up_arch_exit(struct basic_block *bb, struct instruction *ret)
1852 {
1853         pseudo_t pseudo = ret->src;

1855         if (pseudo && pseudo != VOID) {
1856                 struct storage *out = lookup_storage(bb, pseudo, STOR_OUT);
1857                 if (!out) {
1858                         out = alloc_storage();
1859                         add_storage(out, bb, pseudo, STOR_OUT);
1860                 }
1861                 out->type = REG_REG;
1862                 out->regno = 0;
1863         }
1864 }

1866 /*
1867  * Set up dummy/silly output storage information for a switch
1868  * instruction. We need to make sure that a register is available
1869  * when we generate code for switch, so force that by creating
1870  * a dummy output rule.
1871  */
1872 static void set_up_arch_switch(struct basic_block *bb, struct instruction *insn)
1873 {
1874         pseudo_t pseudo = insn->cond;
1875         struct storage *out = lookup_storage(bb, pseudo, STOR_OUT);
1876         if (!out) {
1877                 out = alloc_storage();
1878                 add_storage(out, bb, pseudo, STOR_OUT);
1879         }
1880         out->type = REG_REG;
1881         out->regno = SWITCH_REG;
1882 }

1884 static void arch_set_up_storage(struct entrypoint *ep)
1885 {
1886         struct basic_block *bb;

1888         /* Argument storage etc.. */
1889         set_up_arch_entry(ep, ep->entry);

1891         FOR_EACH_PTR(ep->bbs, bb) {
1892                 struct instruction *insn = last_instruction(bb->insns);
1893                 if (!insn)
1894                         continue;
1895                 switch (insn->opcode) {
1896                 case OP_RET:
1897                         set_up_arch_exit(bb, insn);
1898                         break;
1899                 case OP_SWITCH:
1900                         set_up_arch_switch(bb, insn);
1901                         break;
1902                 default:
1903                         /* nothing */;
1904                 }
1905         } END_FOR_EACH_PTR(bb);
1906 }

1908 static void output(struct entrypoint *ep)
```



```
1909 {
1910         unsigned long generation = ++bb_generation;

1912         last_reg = -1;
1913         stack_offset = 0;

1915         /* Get rid of SSA form (phinodes etc) */
1916         unssa(ep);

1918         /* Set up initial inter-bb storage links */
1919         set_up_storage(ep);

1921         /* Architecture-specific storage rules.. */
1922         arch_set_up_storage(ep);

1924         /* Show the results ... */
1925         output_bb(ep->entry->bb, generation);

1927         /* Clear the storage hashes for the next function.. */
1928         free_storage();
1929 }

1931 static int compile(struct symbol_list *list)
1932 {
1933         struct symbol *sym;
1934         FOR_EACH_PTR(list, sym) {
1935                 struct entrypoint *ep;
1936                 expand_symbol(sym);
1937                 ep = linearize_symbol(sym);
1938                 if (ep)
1939                         output(ep);
1940         } END_FOR_EACH_PTR(sym);
1941
1942         return 0;
1943 }

1945 int main(int argc, char **argv)
1946 {
1947         struct string_list *filelist = NULL;
1948         char *file;

1950         compile(sparse_initialize(argc, argv, &filelist));
1951         dbg_dead = 1;
1952         FOR_EACH_PTR_NOTAG(filelist, file) {
1953                 compile(sparse(file));
1954         } END_FOR_EACH_PTR_NOTAG(file);
1955         return 0;
1956 }
```

```
**********************************************************
   30435 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/expand.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/expand.c
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * expand constant expressions.
 */
#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <limits.h>

#include "lib.h"
#include "allocate.h"
#include "parse.h"
#include "token.h"
#include "symbol.h"
#include "target.h"
#include "expression.h"
#include "expand.h"


static int expand_expression(struct expression *);
static int expand_statement(struct statement *);
static int conservative;

static int expand_symbol_expression(struct expression *expr)
{
        struct symbol *sym = expr->symbol;

        if (sym == &zero_int) {
                if (Wundef)
                        warning(expr->pos, "undefined preprocessor identifier '%
                expr->type = EXPR_VALUE;
                expr->value = 0;
                expr->taint = 0;
```



```
                return 0;
        }
        /* The cost of a symbol expression is lower for on-stack symbols */
        return (sym->ctype.modifiers & (MOD_STATIC | MOD_EXTERN)) ? 2 : 1;
}

static long long get_longlong(struct expression *expr)
{
        int no_expand = expr->ctype->ctype.modifiers & MOD_UNSIGNED;
        long long mask = 1ULL << (expr->ctype->bit_size - 1);
        long long value = expr->value;
        long long ormask, andmask;

        if (!(value & mask))
                no_expand = 1;
        andmask = mask | (mask-1);
        ormask = ~andmask;
        if (no_expand)
                ormask = 0;
        return (value & andmask) | ormask;
}

void cast_value(struct expression *expr, struct symbol *newtype,
                struct expression *old, struct symbol *oldtype)
{
        int old_size = oldtype->bit_size;
        int new_size = newtype->bit_size;
        long long value, mask, signmask;
        long long oldmask, oldsignmask, dropped;

        if (is_float_type(newtype) || is_float_type(oldtype))
                goto Float;

        // For pointers and integers, we can just move the value around
        expr->type = EXPR_VALUE;
        expr->taint = old->taint;
        if (old_size == new_size) {
                expr->value = old->value;
                return;
        }

        // expand it to the full "long long" value
        value = get_longlong(old);

Int:
        // _Bool requires a zero test rather than truncation.
        if (is_bool_type(newtype)) {
                expr->value = !!value;
                if (!conservative && value != 0 && value != 1)
                        warning(old->pos, "odd constant _Bool cast (%llx becomes
                return;
        }

        // Truncate it to the new size
        signmask = 1ULL << (new_size-1);
        mask = signmask | (signmask-1);
        expr->value = value & mask;

        // Stop here unless checking for truncation
        if (!Wcast_truncate || conservative)
                return;

        // Check if we dropped any bits..
        oldsignmask = 1ULL << (old_size-1);
        oldmask = oldsignmask | (oldsignmask-1);
        dropped = oldmask & ~mask;
```

```
128         // OK if the bits were (and still are) purely sign bits
129         if (value & dropped) {
130                 if (!(value & oldsignmask) || !(value & signmask) || (value & dr
131                         warning(old->pos, "cast truncates bits from constant val
132                                 value & oldmask,
133                                 value & mask);
134         }
135         return;

137 Float:
138         if (!is_float_type(newtype)) {
139                 value = (long long)old->fvalue;
140                 expr->type = EXPR_VALUE;
141                 expr->taint = 0;
142                 goto Int;
143         }

145         if (!is_float_type(oldtype))
146                 expr->fvalue = (long double)get_longlong(old);
147         else
148                 expr->fvalue = old->fvalue;

150         if (!(newtype->ctype.modifiers & MOD_LONGLONG) && \
151             !(newtype->ctype.modifiers & MOD_LONGLONGLONG)) {
152                 if ((newtype->ctype.modifiers & MOD_LONG))
153                         expr->fvalue = (double)expr->fvalue;
154                 else
155                         expr->fvalue = (float)expr->fvalue;
156         }
157         expr->type = EXPR_FVALUE;
158 }

160 static int check_shift_count(struct expression *expr, struct symbol *ctype, unsi
161 {
162         warning(expr->pos, "shift too big (%u) for type %s", count, show_typenam
163         count &= ctype->bit_size-1;
164         return count;
165 }

167 /*
168  * CAREFUL! We need to get the size and sign of the
169  * result right!
170  */
171 #define CONVERT(op,s)   (((op)<<1)+(s))
172 #define SIGNED(op)      CONVERT(op, 1)
173 #define UNSIGNED(op)    CONVERT(op, 0)
174 static int simplify_int_binop(struct expression *expr, struct symbol *ctype)
175 {
176         struct expression *left = expr->left, *right = expr->right;
177         unsigned long long v, l, r, mask;
178         signed long long sl, sr;
179         int is_signed;

181         if (right->type != EXPR_VALUE)
182                 return 0;
183         r = right->value;
184         if (expr->op == SPECIAL_LEFTSHIFT || expr->op == SPECIAL_RIGHTSHIFT) {
185                 if (r >= ctype->bit_size) {
186                         if (conservative)
187                                 return 0;
188                         r = check_shift_count(expr, ctype, r);
189                         right->value = r;
190                 }
191         }
192         if (left->type != EXPR_VALUE)
```



```
193                 return 0;
194         l = left->value; r = right->value;
195         is_signed = !(ctype->ctype.modifiers & MOD_UNSIGNED);
196         mask = 1ULL << (ctype->bit_size-1);
197         sl = l; sr = r;
198         if (is_signed && (sl & mask))
199                 sl |= ~(mask-1);
200         if (is_signed && (sr & mask))
201                 sr |= ~(mask-1);
202
203         switch (CONVERT(expr->op,is_signed)) {
204         case SIGNED('+'):
205         case UNSIGNED('+'):
206                 v = l + r;
207                 break;

209         case SIGNED('-'):
210         case UNSIGNED('-'):
211                 v = l - r;
212                 break;

214         case SIGNED('&'):
215         case UNSIGNED('&'):
216                 v = l & r;
217                 break;

219         case SIGNED('|'):
220         case UNSIGNED('|'):
221                 v = l | r;
222                 break;

224         case SIGNED('^'):
225         case UNSIGNED('^'):
226                 v = l ^ r;
227                 break;

229         case SIGNED('*'):
230                 v = sl * sr;
231                 break;

233         case UNSIGNED('*'):
234                 v = l * r;
235                 break;

237         case SIGNED('/'):
238                 if (!r)
239                         goto Div;
240                 if (l == mask && sr == -1)
241                         goto Overflow;
242                 v = sl / sr;
243                 break;

245         case UNSIGNED('/'):
246                 if (!r) goto Div;
247                 v = l / r;
248                 break;

250         case SIGNED('%'):
251                 if (!r)
252                         goto Div;
253                 if (l == mask && sr == -1)
254                         goto Overflow;
255                 v = sl % sr;
256                 break;

258         case UNSIGNED('%'):
```

```
259                 if (!r) goto Div;
260                 v = l % r;
261                 break;

263         case SIGNED(SPECIAL_LEFTSHIFT):
264         case UNSIGNED(SPECIAL_LEFTSHIFT):
265                 v = l << r;
266                 break;

268         case SIGNED(SPECIAL_RIGHTSHIFT):
269                 v = sl >> r;
270                 break;

272         case UNSIGNED(SPECIAL_RIGHTSHIFT):
273                 v = l >> r;
274                 break;

276         default:
277                 return 0;
278         }
279         mask = mask | (mask-1);
280         expr->value = v & mask;
281         expr->type = EXPR_VALUE;
282         expr->taint = left->taint | right->taint;
283         return 1;
284 Div:
285         if (!conservative)
286                 warning(expr->pos, "division by zero");
287         return 0;
288 Overflow:
289         if (!conservative)
290                 warning(expr->pos, "constant integer operation overflow");
291         return 0;
292 }

294 static int simplify_cmp_binop(struct expression *expr, struct symbol *ctype)
295 {
296         struct expression *left = expr->left, *right = expr->right;
297         unsigned long long l, r, mask;
298         signed long long sl, sr;

300         if (left->type != EXPR_VALUE || right->type != EXPR_VALUE)
301                 return 0;
302         l = left->value; r = right->value;
303         mask = 1ULL << (ctype->bit_size-1);
304         sl = l; sr = r;
305         if (sl & mask)
306                 sl |= ~(mask-1);
307         if (sr & mask)
308                 sr |= ~(mask-1);
309         switch (expr->op) {
310         case '<':               expr->value = sl < sr; break;
311         case '>':               expr->value = sl > sr; break;
312         case SPECIAL_LTE:       expr->value = sl <= sr; break;
313         case SPECIAL_GTE:       expr->value = sl >= sr; break;
314         case SPECIAL_EQUAL:     expr->value = l == r; break;
315         case SPECIAL_NOTEQUAL:  expr->value = l != r; break;
316         case SPECIAL_UNSIGNED_LT:expr->value = l < r; break;
317         case SPECIAL_UNSIGNED_GT:expr->value = l > r; break;
318         case SPECIAL_UNSIGNED_LTE:expr->value = l <= r; break;
319         case SPECIAL_UNSIGNED_GTE:expr->value = l >= r; break;
320         }
321         expr->type = EXPR_VALUE;
322         expr->taint = left->taint | right->taint;
323         return 1;
324 }
```



```
326 static int simplify_float_binop(struct expression *expr)
327 {
328         struct expression *left = expr->left, *right = expr->right;
329         unsigned long mod = expr->ctype->ctype.modifiers;
330         long double l, r, res;

332         if (left->type != EXPR_FVALUE || right->type != EXPR_FVALUE)
333                 return 0;

335         l = left->fvalue;
336         r = right->fvalue;

338         if (mod & MOD_LONGLONG) {
339                 switch (expr->op) {
340                 case '+':       res = l + r; break;
341                 case '-':       res = l - r; break;
342                 case '*':       res = l * r; break;
343                 case '/':       if (!r) goto Div;
344                                 res = l / r; break;
345                 default: return 0;
346                 }
347         } else if (mod & MOD_LONG) {
348                 switch (expr->op) {
349                 case '+':       res = (double) l + (double) r; break;
350                 case '-':       res = (double) l - (double) r; break;
351                 case '*':       res = (double) l * (double) r; break;
352                 case '/':       if (!r) goto Div;
353                                 res = (double) l / (double) r; break;
354                 default: return 0;
355                 }
356         } else {
357                 switch (expr->op) {
358                 case '+':       res = (float)l + (float)r; break;
359                 case '-':       res = (float)l - (float)r; break;
360                 case '*':       res = (float)l * (float)r; break;
361                 case '/':       if (!r) goto Div;
362                                 res = (float)l / (float)r; break;
363                 default: return 0;
364                 }
365         }
366         expr->type = EXPR_FVALUE;
367         expr->fvalue = res;
368         return 1;
369 Div:
370         if (!conservative)
371                 warning(expr->pos, "division by zero");
372         return 0;
373 }

375 static int simplify_float_cmp(struct expression *expr, struct symbol *ctype)
376 {
377         struct expression *left = expr->left, *right = expr->right;
378         long double l, r;

380         if (left->type != EXPR_FVALUE || right->type != EXPR_FVALUE)
381                 return 0;

383         l = left->fvalue;
384         r = right->fvalue;
385         switch (expr->op) {
386         case '<':               expr->value = l < r; break;
387         case '>':               expr->value = l > r; break;
388         case SPECIAL_LTE:       expr->value = l <= r; break;
389         case SPECIAL_GTE:       expr->value = l >= r; break;
390         case SPECIAL_EQUAL:     expr->value = l == r; break;
```

```
391         case SPECIAL_NOTEQUAL:  expr->value = l != r; break;
392         }
393         expr->type = EXPR_VALUE;
394         expr->taint = 0;
395         return 1;
396 }

398 static int expand_binop(struct expression *expr)
399 {
400         int cost;

402         cost = expand_expression(expr->left);
403         cost += expand_expression(expr->right);
404         if (simplify_int_binop(expr, expr->ctype))
405                 return 0;
406         if (simplify_float_binop(expr))
407                 return 0;
408         return cost + 1;
409 }

411 static int expand_logical(struct expression *expr)
412 {
413         struct expression *left = expr->left;
414         struct expression *right;
415         int cost, rcost;

417         /* Do immediate short-circuiting ... */
418         cost = expand_expression(left);
419         if (left->type == EXPR_VALUE) {
420                 if (expr->op == SPECIAL_LOGICAL_AND) {
421                         if (!left->value) {
422                                 expr->type = EXPR_VALUE;
423                                 expr->value = 0;
424                                 expr->taint = left->taint;
425                                 return 0;
426                         }
427                 } else {
428                         if (left->value) {
429                                 expr->type = EXPR_VALUE;
430                                 expr->value = 1;
431                                 expr->taint = left->taint;
432                                 return 0;
433                         }
434                 }
435         }

437         right = expr->right;
438         rcost = expand_expression(right);
439         if (left->type == EXPR_VALUE && right->type == EXPR_VALUE) {
440                 /*
441                  * We know the left value doesn't matter, since
442                  * otherwise we would have short-circuited it..
443                  */
444                 expr->type = EXPR_VALUE;
445                 expr->value = right->value != 0;
446                 expr->taint = left->taint | right->taint;
447                 return 0;
448         }

450         /*
451          * If the right side is safe and cheaper than a branch,
452          * just avoid the branch and turn it into a regular binop
453          * style SAFELOGICAL.
454          */
455         if (rcost < BRANCH_COST) {
456                 expr->type = EXPR_BINOP;
```



```
457                 rcost -= BRANCH_COST - 1;
458         }

460         return cost + BRANCH_COST + rcost;
461 }

463 static int expand_comma(struct expression *expr)
464 {
465         int cost;

467         cost = expand_expression(expr->left);
468         cost += expand_expression(expr->right);
469         if (expr->left->type == EXPR_VALUE || expr->left->type == EXPR_FVALUE) {
470                 unsigned flags = expr->flags;
471                 unsigned taint;
472                 taint = expr->left->type == EXPR_VALUE ? expr->left->taint : 0;
473                 *expr = *expr->right;
474                 expr->flags = flags;
475                 if (expr->type == EXPR_VALUE)
476                         expr->taint |= Taint_comma | taint;
477         }
478         return cost;
479 }

481 #define MOD_IGN (MOD_VOLATILE | MOD_CONST)

483 static int compare_types(int op, struct symbol *left, struct symbol *right)
484 {
485         struct ctype c1 = {.base_type = left};
486         struct ctype c2 = {.base_type = right};
487         switch (op) {
488         case SPECIAL_EQUAL:
489                 return !type_difference(&c1, &c2, MOD_IGN, MOD_IGN);
490         case SPECIAL_NOTEQUAL:
491                 return type_difference(&c1, &c2, MOD_IGN, MOD_IGN) != NULL;
492         case '<':
493                 return left->bit_size < right->bit_size;
494         case '>':
495                 return left->bit_size > right->bit_size;
496         case SPECIAL_LTE:
497                 return left->bit_size <= right->bit_size;
498         case SPECIAL_GTE:
499                 return left->bit_size >= right->bit_size;
500         }
501         return 0;
502 }

504 static int expand_compare(struct expression *expr)
505 {
506         struct expression *left = expr->left, *right = expr->right;
507         int cost;

509         cost = expand_expression(left);
510         cost += expand_expression(right);

512         if (left && right) {
513                 /* Type comparison? */
514                 if (left->type == EXPR_TYPE && right->type == EXPR_TYPE) {
515                         int op = expr->op;
516                         expr->type = EXPR_VALUE;
517                         expr->value = compare_types(op, left->symbol, right->sym
518                         expr->taint = 0;
519                         return 0;
520                 }
521                 if (simplify_cmp_binop(expr, left->ctype))
522                         return 0;
```

```
523                 if (simplify_float_cmp(expr, left->ctype))
524                         return 0;
525         }
526         return cost + 1;
527 }

529 static int expand_conditional(struct expression *expr)
530 {
531         struct expression *cond = expr->conditional;
532         struct expression *true = expr->cond_true;
533         struct expression *false = expr->cond_false;
534         int cost, cond_cost;

536         cond_cost = expand_expression(cond);
537         if (cond->type == EXPR_VALUE) {
538                 unsigned flags = expr->flags;
539                 if (!cond->value)
540                         true = false;
541                 if (!true)
542                         true = cond;
543                 cost = expand_expression(true);
544                 *expr = *true;
545                 expr->flags = flags;
546                 if (expr->type == EXPR_VALUE)
547                         expr->taint |= cond->taint;
548                 return cost;
549         }

551         cost = expand_expression(true);
552         cost += expand_expression(false);

554         if (cost < SELECT_COST) {
555                 expr->type = EXPR_SELECT;
556                 cost -= BRANCH_COST - 1;
557         }

559         return cost + cond_cost + BRANCH_COST;
560 }
561
562 static int expand_assignment(struct expression *expr)
563 {
564         expand_expression(expr->left);
565         expand_expression(expr->right);
566         return SIDE_EFFECTS;
567 }

569 static int expand_addressof(struct expression *expr)
570 {
571         return expand_expression(expr->unop);
572 }

574 /*
575  * Look up a trustable initializer value at the requested offset.
576  *
577  * Return NULL if no such value can be found or statically trusted.
578  *
579  * FIXME!! We should check that the size is right!
580  */
581 static struct expression *constant_symbol_value(struct symbol *sym, int offset)
582 {
583         struct expression *value;

585         if (sym->ctype.modifiers & (MOD_ASSIGNED | MOD_ADDRESSABLE))
586                 return NULL;
587         value = sym->initializer;
588         if (!value)
```



```
589                 return NULL;
590         if (value->type == EXPR_INITIALIZER) {
591                 struct expression *entry;
592                 FOR_EACH_PTR(value->expr_list, entry) {
593                         if (entry->type != EXPR_POS) {
594                                 if (offset)
595                                         continue;
596                                 return entry;
597                         }
598                         if (entry->init_offset < offset)
599                                 continue;
600                         if (entry->init_offset > offset)
601                                 return NULL;
602                         return entry->init_expr;
603                 } END_FOR_EACH_PTR(entry);
604                 return NULL;
605         }
606         return value;
607 }

609 static int expand_dereference(struct expression *expr)
610 {
611         struct expression *unop = expr->unop;
612         unsigned int offset;

614         expand_expression(unop);

616         /*
617          * NOTE! We get a bogus warning right now for some special
618          * cases: apparently I've screwed up the optimization of
619          * a zero-offset dereference, and the ctype is wrong.
620          *
621          * Leave the warning in anyway, since this is also a good
622          * test for me to get the type evaluation right..
623          */
624         if (expr->ctype->ctype.modifiers & MOD_NODEREF)
625                 warning(unop->pos, "dereference of noderef expression");

627         /*
628          * Is it "symbol" or "symbol + offset"?
629          */
630         offset = 0;
631         if (unop->type == EXPR_BINOP && unop->op == '+') {
632                 struct expression *right = unop->right;
633                 if (right->type == EXPR_VALUE) {
634                         offset = right->value;
635                         unop = unop->left;
636                 }
637         }

639         if (unop->type == EXPR_SYMBOL) {
640                 struct symbol *sym = unop->symbol;
641                 struct expression *value = constant_symbol_value(sym, offset);

643                 /* Const symbol with a constant initializer? */
644                 if (value) {
645                         /* FIXME! We should check that the size is right! */
646                         if (value->type == EXPR_VALUE) {
647                                 expr->type = EXPR_VALUE;
648                                 expr->value = value->value;
649                                 expr->taint = 0;
650                                 return 0;
651                         } else if (value->type == EXPR_FVALUE) {
652                                 expr->type = EXPR_FVALUE;
653                                 expr->fvalue = value->fvalue;
654                                 return 0;
```

```
655                         }
656                 }

658                 /* Direct symbol dereference? Cheap and safe */
659                 return (sym->ctype.modifiers & (MOD_STATIC | MOD_EXTERN)) ? 2 :
660         }

662         return UNSAFE;
663 }

665 static int simplify_preop(struct expression *expr)
666 {
667         struct expression *op = expr->unop;
668         unsigned long long v, mask;

670         if (op->type != EXPR_VALUE)
671                 return 0;

673         mask = 1ULL << (expr->ctype->bit_size-1);
674         v = op->value;
675         switch (expr->op) {
676         case '+': break;
677         case '-':
678                 if (v == mask && !(expr->ctype->ctype.modifiers & MOD_UNSIGNED))
679                         goto Overflow;
680                 v = -v;
681                 break;
682         case '!': v = !v; break;
683         case '~': v = ~v; break;
684         default: return 0;
685         }
686         mask = mask | (mask-1);
687         expr->value = v & mask;
688         expr->type = EXPR_VALUE;
689         expr->taint = op->taint;
690         return 1;

692 Overflow:
693         if (!conservative)
694                 warning(expr->pos, "constant integer operation overflow");
695         return 0;
696 }

698 static int simplify_float_preop(struct expression *expr)
699 {
700         struct expression *op = expr->unop;
701         long double v;

703         if (op->type != EXPR_FVALUE)
704                 return 0;
705         v = op->fvalue;
706         switch (expr->op) {
707         case '+': break;
708         case '-': v = -v; break;
709         default: return 0;
710         }
711         expr->fvalue = v;
712         expr->type = EXPR_FVALUE;
713         return 1;
714 }

716 /*
717  * Unary post-ops: x++ and x--
718  */
719 static int expand_postop(struct expression *expr)
720 {
```



```
721         expand_expression(expr->unop);
722         return SIDE_EFFECTS;
723 }

725 static int expand_preop(struct expression *expr)
726 {
727         int cost;

729         switch (expr->op) {
730         case '*':
731                 return expand_dereference(expr);

733         case '&':
734                 return expand_addressof(expr);

736         case SPECIAL_INCREMENT:
737         case SPECIAL_DECREMENT:
738                 /*
739                  * From a type evaluation standpoint the preops are
740                  * the same as the postops
741                  */
742                 return expand_postop(expr);

744         default:
745                 break;
746         }
747         cost = expand_expression(expr->unop);

749         if (simplify_preop(expr))
750                 return 0;
751         if (simplify_float_preop(expr))
752                 return 0;
753         return cost + 1;
754 }

756 static int expand_arguments(struct expression_list *head)
757 {
758         int cost = 0;
759         struct expression *expr;

761         FOR_EACH_PTR (head, expr) {
762                 cost += expand_expression(expr);
763         } END_FOR_EACH_PTR(expr);
764         return cost;
765 }

767 static int expand_cast(struct expression *expr)
768 {
769         int cost;
770         struct expression *target = expr->cast_expression;

772         cost = expand_expression(target);

774         /* Simplify normal integer casts.. */
775         if (target->type == EXPR_VALUE || target->type == EXPR_FVALUE) {
776                 cast_value(expr, expr->ctype, target, target->ctype);
777                 return 0;
778         }
779         return cost + 1;
780 }

782 /*
783  * expand a call expression with a symbol. This
784  * should expand builtins.
785  */
786 static int expand_symbol_call(struct expression *expr, int cost)
```

```
787 {
788         struct expression *fn = expr->fn;
789         struct symbol *ctype = fn->ctype;
790
791         if (fn->type != EXPR_PREOP)
792                 return SIDE_EFFECTS;
793
794         if (ctype->op && ctype->op->expand)
795                 return ctype->op->expand(expr, cost);
796
797         if (ctype->ctype.modifiers & MOD_PURE)
798                 return cost + 1;
799
800         return SIDE_EFFECTS;
801 }
802
803 static int expand_call(struct expression *expr)
804 {
805         int cost;
806         struct symbol *sym;
807         struct expression *fn = expr->fn;
808
809         cost = expand_arguments(expr->args);
810         sym = fn->ctype;
811         if (!sym) {
812                 expression_error(expr, "function has no type");
813                 return SIDE_EFFECTS;
814         }
815         if (sym->type == SYM_NODE)
816                 return expand_symbol_call(expr, cost);
817
818         return SIDE_EFFECTS;
819 }
820
821 static int expand_expression_list(struct expression_list *list)
822 {
823         int cost = 0;
824         struct expression *expr;
825
826         FOR_EACH_PTR(list, expr) {
827                 cost += expand_expression(expr);
828         } END_FOR_EACH_PTR(expr);
829         return cost;
830 }
831
832 /*
833  * We can simplify nested position expressions if
834  * this is a simple (single) positional expression.
835  */
836 static int expand_pos_expression(struct expression *expr)
837 {
838         struct expression *nested = expr->init_expr;
839         unsigned long offset = expr->init_offset;
840         int nr = expr->init_nr;
841
842         if (nr == 1) {
843                 switch (nested->type) {
844                 case EXPR_POS:
845                         offset += nested->init_offset;
846                         *expr = *nested;
847                         expr->init_offset = offset;
848                         nested = expr;
849                         break;
850
851                 case EXPR_INITIALIZER: {
852                         struct expression *reuse = nested, *entry;
```



```
853                         *expr = *nested;
854                         FOR_EACH_PTR(expr->expr_list, entry) {
855                                 if (entry->type == EXPR_POS) {
856                                         entry->init_offset += offset;
857                                 } else {
858                                         if (!reuse) {
859                                                 /*
860                                                  * This happens rarely, but it c
861                                                  * with bitfields that are all a
862                                                  * zero..
863                                                  */
864                                                 reuse = alloc_expression(entry->
865                                         }
866                                         reuse->type = EXPR_POS;
867                                         reuse->ctype = entry->ctype;
868                                         reuse->init_offset = offset;
869                                         reuse->init_nr = 1;
870                                         reuse->init_expr = entry;
871                                         REPLACE_CURRENT_PTR(entry, reuse);
872                                         reuse = NULL;
873                                 }
874                         } END_FOR_EACH_PTR(entry);
875                         nested = expr;
876                         break;
877                 }
878
879                 default:
880                         break;
881                 }
882         }
883         return expand_expression(nested);
884 }
885
886 static unsigned long bit_offset(const struct expression *expr)
887 {
888         unsigned long offset = 0;
889         while (expr->type == EXPR_POS) {
890                 offset += bytes_to_bits(expr->init_offset);
891                 expr = expr->init_expr;
892         }
893         if (expr && expr->ctype)
894                 offset += expr->ctype->bit_offset;
895         return offset;
896 }
897
898 static unsigned long bit_range(const struct expression *expr)
899 {
900         unsigned long range = 0;
901         unsigned long size = 0;
902         while (expr->type == EXPR_POS) {
903                 unsigned long nr = expr->init_nr;
904                 size = expr->ctype->bit_size;
905                 range += (nr - 1) * size;
906                 expr = expr->init_expr;
907         }
908         range += size;
909         return range;
910 }
911
912 static int compare_expressions(const void *_a, const void *_b)
913 {
914         const struct expression *a = _a;
915         const struct expression *b = _b;
916         unsigned long a_pos = bit_offset(a);
917         unsigned long b_pos = bit_offset(b);
```

```
919         return (a_pos < b_pos) ? -1 : (a_pos == b_pos) ? 0 : 1;
920 }

922 static void sort_expression_list(struct expression_list **list)
923 {
924         sort_list((struct ptr_list **)list, compare_expressions);
925 }

927 static void verify_nonoverlapping(struct expression_list **list, struct expressi
928 {
929         struct expression *a = NULL;
930         unsigned long max = 0;
931         unsigned long whole = expr->ctype->bit_size;
932         struct expression *b;

934         if (!Woverride_init)
935                 return;

937         FOR_EACH_PTR(*list, b) {
938                 unsigned long off, end;
939                 if (!b->ctype || !b->ctype->bit_size)
940                         continue;
941                 off = bit_offset(b);
942                 if (a && off < max) {
943                         warning(a->pos, "Initializer entry defined twice");
944                         info(b->pos, "  also defined here");
945                         if (!Woverride_init_all)
946                                 return;
947                 }
948                 end = off + bit_range(b);
949                 if (!a && !Woverride_init_whole_range) {
950                         // If first entry is the whole range, do not let
951                         // any warning about it (this allow to initialize
952                         // an array with some default value and then override
953                         // some specific entries).
954                         if (off == 0 && end == whole)
955                                 continue;
956                 }
957                 if (end > max) {
958                         max = end;
959                         a = b;
960                 }
961         } END_FOR_EACH_PTR(b);
962 }

964 static int expand_expression(struct expression *expr)
965 {
966         if (!expr)
967                 return 0;
968         if (!expr->ctype || expr->ctype == &bad_ctype)
969                 return UNSAFE;

971         switch (expr->type) {
972         case EXPR_VALUE:
973         case EXPR_FVALUE:
974         case EXPR_STRING:
975                 return 0;
976         case EXPR_TYPE:
977         case EXPR_SYMBOL:
978                 return expand_symbol_expression(expr);
979         case EXPR_BINOP:
980                 return expand_binop(expr);

982         case EXPR_LOGICAL:
983                 return expand_logical(expr);
```



```
985         case EXPR_COMMA:
986                 return expand_comma(expr);

988         case EXPR_COMPARE:
989                 return expand_compare(expr);

991         case EXPR_ASSIGNMENT:
992                 return expand_assignment(expr);

994         case EXPR_PREOP:
995                 return expand_preop(expr);

997         case EXPR_POSTOP:
998                 return expand_postop(expr);

1000        case EXPR_CAST:
1001        case EXPR_FORCE_CAST:
1002        case EXPR_IMPLIED_CAST:
1003                return expand_cast(expr);

1005        case EXPR_CALL:
1006                return expand_call(expr);

1008        case EXPR_DEREF:
1009                warning(expr->pos, "we should not have an EXPR_DEREF left at exp
1010                return UNSAFE;

1012        case EXPR_SELECT:
1013        case EXPR_CONDITIONAL:
1014                return expand_conditional(expr);

1016        case EXPR_STATEMENT: {
1017                struct statement *stmt = expr->statement;
1018                int cost = expand_statement(stmt);

1020                if (stmt->type == STMT_EXPRESSION && stmt->expression)
1021                        *expr = *stmt->expression;
1022                return cost;
1023        }

1025        case EXPR_LABEL:
1026                return 0;

1028        case EXPR_INITIALIZER:
1029                sort_expression_list(&expr->expr_list);
1030                verify_nonoverlapping(&expr->expr_list, expr);
1031                return expand_expression_list(expr->expr_list);

1033        case EXPR_IDENTIFIER:
1034                return UNSAFE;

1036        case EXPR_INDEX:
1037                return UNSAFE;

1039        case EXPR_SLICE:
1040                return expand_expression(expr->base) + 1;

1042        case EXPR_POS:
1043                return expand_pos_expression(expr);

1045        case EXPR_SIZEOF:
1046        case EXPR_PTRSIZEOF:
1047        case EXPR_ALIGNOF:
1048        case EXPR_OFFSETOF:
1049                expression_error(expr, "internal front-end error: sizeof in expa
1050                return UNSAFE;
```

```
1051         }
1052         return SIDE_EFFECTS;
1053 }

1055 static void expand_const_expression(struct expression *expr, const char *where)
1056 {
1057         if (expr) {
1058                 expand_expression(expr);
1059                 if (expr->type != EXPR_VALUE)
1060                         expression_error(expr, "Expected constant expression in
1061         }
1062 }

1064 int expand_symbol(struct symbol *sym)
1065 {
1066         int retval;
1067         struct symbol *base_type;

1069         if (!sym)
1070                 return 0;
1071         base_type = sym->ctype.base_type;
1072         if (!base_type)
1073                 return 0;

1075         retval = expand_expression(sym->initializer);
1076         /* expand the body of the symbol */
1077         if (base_type->type == SYM_FN) {
1078                 if (base_type->stmt)
1079                         expand_statement(base_type->stmt);
1080         }
1081         return retval;
1082 }

1084 static void expand_return_expression(struct statement *stmt)
1085 {
1086         expand_expression(stmt->expression);
1087 }

1089 static int expand_if_statement(struct statement *stmt)
1090 {
1091         struct expression *expr = stmt->if_conditional;

1093         if (!expr || !expr->ctype || expr->ctype == &bad_ctype)
1094                 return UNSAFE;

1096         expand_expression(expr);

1098 /* This is only valid if nobody jumps into the "dead" side */
1099 #if 0
1100         /* Simplify constant conditionals without even evaluating the false side
1101         if (expr->type == EXPR_VALUE) {
1102                 struct statement *simple;
1103                 simple = expr->value ? stmt->if_true : stmt->if_false;

1105                 /* Nothing? */
1106                 if (!simple) {
1107                         stmt->type = STMT_NONE;
1108                         return 0;
1109                 }
1110                 expand_statement(simple);
1111                 *stmt = *simple;
1112                 return SIDE_EFFECTS;
1113         }
1114 #endif
1115         expand_statement(stmt->if_true);
1116         expand_statement(stmt->if_false);
```



```
1117         return SIDE_EFFECTS;
1118 }

1120 /*
1121  * Expanding a compound statement is really just
1122  * about adding up the costs of each individual
1123  * statement.
1124  *
1125  * We also collapse a simple compound statement:
1126  * this would trigger for simple inline functions,
1127  * except we would have to check the "return"
1128  * symbol usage. Next time.
1129  */
1130 static int expand_compound(struct statement *stmt)
1131 {
1132         struct statement *s, *last;
1133         int cost, statements;

1135         if (stmt->ret)
1136                 expand_symbol(stmt->ret);

1138         last = stmt->args;
1139         cost = expand_statement(last);
1140         statements = last != NULL;
1141         FOR_EACH_PTR(stmt->stmts, s) {
1142                 statements++;
1143                 last = s;
1144                 cost += expand_statement(s);
1145         } END_FOR_EACH_PTR(s);

1147         if (statements == 1 && !stmt->ret)
1148                 *stmt = *last;

1150         return cost;
1151 }

1153 static int expand_statement(struct statement *stmt)
1154 {
1155         if (!stmt)
1156                 return 0;

1158         switch (stmt->type) {
1159         case STMT_DECLARATION: {
1160                 struct symbol *sym;
1161                 FOR_EACH_PTR(stmt->declaration, sym) {
1162                         expand_symbol(sym);
1163                 } END_FOR_EACH_PTR(sym);
1164                 return SIDE_EFFECTS;
1165         }

1167         case STMT_RETURN:
1168                 expand_return_expression(stmt);
1169                 return SIDE_EFFECTS;

1171         case STMT_EXPRESSION:
1172                 return expand_expression(stmt->expression);

1174         case STMT_COMPOUND:
1175                 return expand_compound(stmt);

1177         case STMT_IF:
1178                 return expand_if_statement(stmt);

1180         case STMT_ITERATOR:
1181                 expand_expression(stmt->iterator_pre_condition);
1182                 expand_expression(stmt->iterator_post_condition);
```

```
1183                 expand_statement(stmt->iterator_pre_statement);
1184                 expand_statement(stmt->iterator_statement);
1185                 expand_statement(stmt->iterator_post_statement);
1186                 return SIDE_EFFECTS;

1188         case STMT_SWITCH:
1189                 expand_expression(stmt->switch_expression);
1190                 expand_statement(stmt->switch_statement);
1191                 return SIDE_EFFECTS;

1193         case STMT_CASE:
1194                 expand_const_expression(stmt->case_expression, "case statement")
1195                 expand_const_expression(stmt->case_to, "case statement");
1196                 expand_statement(stmt->case_statement);
1197                 return SIDE_EFFECTS;

1199         case STMT_LABEL:
1200                 expand_statement(stmt->label_statement);
1201                 return SIDE_EFFECTS;

1203         case STMT_GOTO:
1204                 expand_expression(stmt->goto_expression);
1205                 return SIDE_EFFECTS;

1207         case STMT_NONE:
1208                 break;
1209         case STMT_ASM:
1210                 /* FIXME! Do the asm parameter evaluation! */
1211                 break;
1212         case STMT_CONTEXT:
1213                 expand_expression(stmt->expression);
1214                 break;
1215         case STMT_RANGE:
1216                 expand_expression(stmt->range_expression);
1217                 expand_expression(stmt->range_low);
1218                 expand_expression(stmt->range_high);
1219                 break;
1220         }
1221         return SIDE_EFFECTS;
1222 }

1224 static inline int bad_integer_constant_expression(struct expression *expr)
1225 {
1226         if (!(expr->flags & CEF_ICE))
1227                 return 1;
1228         if (expr->taint & Taint_comma)
1229                 return 1;
1230         return 0;
1231 }

1233 static long long __get_expression_value(struct expression *expr, int strict)
1234 {
1235         long long value, mask;
1236         struct symbol *ctype;

1238         if (!expr)
1239                 return 0;
1240         ctype = evaluate_expression(expr);
1241         if (!ctype) {
1242                 expression_error(expr, "bad constant expression type");
1243                 return 0;
1244         }
1245         expand_expression(expr);
1246         if (expr->type != EXPR_VALUE) {
1247                 if (strict != 2)
1248                         expression_error(expr, "bad constant expression");
```



```
1249                 return 0;
1250         }
1251         if ((strict == 1) && bad_integer_constant_expression(expr)) {
1252                 expression_error(expr, "bad integer constant expression");
1253                 return 0;
1254         }

1256         value = expr->value;
1257         mask = 1ULL << (ctype->bit_size-1);

1259         if (value & mask) {
1260                 while (ctype->type != SYM_BASETYPE)
1261                         ctype = ctype->ctype.base_type;
1262                 if (!(ctype->ctype.modifiers & MOD_UNSIGNED))
1263                         value = value | mask | ~(mask-1);
1264         }
1265         return value;
1266 }

1268 long long get_expression_value(struct expression *expr)
1269 {
1270         return __get_expression_value(expr, 0);
1271 }

1273 long long const_expression_value(struct expression *expr)
1274 {
1275         return __get_expression_value(expr, 1);
1276 }

1278 long long get_expression_value_silent(struct expression *expr)
1279 {

1281         return __get_expression_value(expr, 2);
1282 }

1284 int expr_truth_value(struct expression *expr)
1285 {
1286         const int saved = conservative;
1287         struct symbol *ctype;

1289         if (!expr)
1290                 return 0;

1292         ctype = evaluate_expression(expr);
1293         if (!ctype)
1294                 return -1;

1296         conservative = 1;
1297         expand_expression(expr);
1298         conservative = saved;

1300 redo:
1301         switch (expr->type) {
1302         case EXPR_COMMA:
1303                 expr = expr->right;
1304                 goto redo;
1305         case EXPR_VALUE:
1306                 return expr->value != 0;
1307         case EXPR_FVALUE:
1308                 return expr->fvalue != 0;
1309         default:
1310                 return -1;
1311         }
1312 }

1314 int is_zero_constant(struct expression *expr)
```

```
1315 {
1316         const int saved = conservative;
1317         conservative = 1;
1318         expand_expression(expr);
1319         conservative = saved;
1320         return expr->type == EXPR_VALUE && !expr->value;
1321 }
```

```
**********************************************************
    1536 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/expand.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef EXPAND_H
#define EXPAND_H
/*
 * sparse/expand.h
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

/* Random cost numbers */
#define SIDE_EFFECTS 10000      /* The expression has side effects */
#define UNSAFE 100              /* The expression may be "infinitely costly" due
#define SELECT_COST 20          /* Cut-off for turning a conditional into a sele
#define BRANCH_COST 10          /* Cost of a conditional branch */

#endif
```

```
**********************************************************
   24723 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/expression.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/expression.c
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * This is the expression parsing part of parsing C.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <limits.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "scope.h"
#include "expression.h"
#include "target.h"
#include "char.h"

static int match_oplist(int op, ...)
{
        va_list args;
        int nextop;

        va_start(args, op);
        do {
                nextop = va_arg(args, int);
        } while (nextop != 0 && nextop != op);
        va_end(args);

        return nextop != 0;
}
```



```
static struct token *comma_expression(struct token *, struct expression **);

struct token *parens_expression(struct token *token, struct expression **expr, c
{
        token = expect(token, '(', where);
        if (match_op(token, '{')) {
                struct expression *e = alloc_expression(token->pos, EXPR_STATEME
                struct statement *stmt = alloc_statement(token->pos, STMT_COMPOU
                *expr = e;
                e->statement = stmt;
                start_symbol_scope(e->pos);
                token = compound_statement(token->next, stmt);
                end_symbol_scope();
                token = expect(token, '}', "at end of statement expression");
        } else
                token = parse_expression(token, expr);
        return expect(token, ')', where);
}

/*
 * Handle __func__, __FUNCTION__ and __PRETTY_FUNCTION__ token
 * conversion
 */
static struct symbol *handle_func(struct token *token)
{
        struct ident *ident = token->ident;
        struct symbol *decl, *array;
        struct string *string;
        int len;

        if (ident != &__func___ident &&
            ident != &__FUNCTION___ident &&
            ident != &__PRETTY_FUNCTION___ident)
                return NULL;

        if (!current_fn || !current_fn->ident)
                return NULL;

        /* OK, it's one of ours */
        array = alloc_symbol(token->pos, SYM_ARRAY);
        array->ctype.base_type = &char_ctype;
        array->ctype.alignment = 1;
        array->endpos = token->pos;
        decl = alloc_symbol(token->pos, SYM_NODE);
        decl->ctype.base_type = array;
        decl->ctype.alignment = 1;
        decl->ctype.modifiers = MOD_STATIC;
        decl->endpos = token->pos;

        /* function-scope, but in NS_SYMBOL */
        bind_symbol(decl, ident, NS_LABEL);
        decl->namespace = NS_SYMBOL;

        len = current_fn->ident->len;
        string = __alloc_string(len + 1);
        memcpy(string->data, current_fn->ident->name, len);
        string->data[len] = 0;
        string->length = len + 1;

        decl->initializer = alloc_expression(token->pos, EXPR_STRING);
        decl->initializer->string = string;
        decl->initializer->ctype = decl;
        decl->array_size = alloc_const_expression(token->pos, len + 1);
        array->array_size = decl->array_size;
        decl->bit_size = array->bit_size = bytes_to_bits(len + 1);
```

```
127         return decl;
128 }

130 static struct token *parse_type(struct token *token, struct expression **tree)
131 {
132         struct symbol *sym;
133         *tree = alloc_expression(token->pos, EXPR_TYPE);
134         token = typename(token, &sym, NULL);
135         if (sym->ident)
136                 sparse_error(token->pos,
137                              "type expression should not include identifier "
138                              "\"%s\"", sym->ident->name);
139         (*tree)->symbol = sym;
140         return token;
141 }

143 static struct token *builtin_types_compatible_p_expr(struct token *token,
144                                                      struct expression **tree)
145 {
146         struct expression *expr = alloc_expression(
147                 token->pos, EXPR_COMPARE);
148         expr->op = SPECIAL_EQUAL;
149         token = token->next;
150         if (!match_op(token, '('))
151                 return expect(token, '(',
152                               "after __builtin_types_compatible_p");
153         token = token->next;
154         token = parse_type(token, &expr->left);
155         if (!match_op(token, ','))
156                 return expect(token, ',',
157                               "in __builtin_types_compatible_p");
158         token = token->next;
159         token = parse_type(token, &expr->right);
160         if (!match_op(token, ')'))
161                 return expect(token, ')',
162                               "at end of __builtin_types_compatible_p");
163         token = token->next;
164
165         *tree = expr;
166         return token;
167 }

169 static struct token *builtin_offsetof_expr(struct token *token,
170                                            struct expression **tree)
171 {
172         struct expression *expr = NULL;
173         struct expression **p = &expr;
174         struct symbol *sym;
175         int op = '.';

177         token = token->next;
178         if (!match_op(token, '('))
179                 return expect(token, '(', "after __builtin_offset");

181         token = token->next;
182         token = typename(token, &sym, NULL);
183         if (sym->ident)
184                 sparse_error(token->pos,
185                              "type expression should not include identifier "
186                              "\"%s\"", sym->ident->name);

188         if (!match_op(token, ','))
189                 return expect(token, ',', "in __builtin_offset");

191         while (1) {
192                 struct expression *e;
```



```
193                 switch (op) {
194                 case ')':
195                         expr->in = sym;
196                         *tree = expr;
197                 default:
198                         return expect(token, ')', "at end of __builtin_offset");
199                 case SPECIAL_DEREFERENCE:
200                         e = alloc_expression(token->pos, EXPR_OFFSETOF);
201                         e->op = '[';
202                         *p = e;
203                         p = &e->down;
204                         /* fall through */
205                 case '.':
206                         token = token->next;
207                         e = alloc_expression(token->pos, EXPR_OFFSETOF);
208                         e->op = '.';
209                         if (token_type(token) != TOKEN_IDENT) {
210                                 sparse_error(token->pos, "Expected member name")
211                                 return token;
212                         }
213                         e->ident = token->ident;
214                         token = token->next;
215                         break;
216                 case '[':
217                         token = token->next;
218                         e = alloc_expression(token->pos, EXPR_OFFSETOF);
219                         e->op = '[';
220                         token = parse_expression(token, &e->index);
221                         token = expect(token, ']',
222                                        "at end of array dereference");
223                         if (!e->index)
224                                 return token;
225                 }
226                 *p = e;
227                 p = &e->down;
228                 op = token_type(token) == TOKEN_SPECIAL ? token->special : 0;
229         }
230 }

232 #ifndef ULLONG_MAX
233 #define ULLONG_MAX (~0ULL)
234 #endif

236 static unsigned long long parse_num(const char *nptr, char **end)
237 {
238         if (nptr[0] == '0' && tolower((unsigned char)nptr[1]) == 'b')
239                 return strtoull(&nptr[2], end, 2);
240         return strtoull(nptr, end, 0);
241 }

243 static void get_number_value(struct expression *expr, struct token *token)
244 {
245         const char *str = token->number;
246         unsigned long long value;
247         char *end;
248         int size = 0, want_unsigned = 0;
249         int overflow = 0, do_warn = 0;
250         int try_unsigned = 1;
251         int bits;

253         errno = 0;
254         value = parse_num(str, &end);
255         if (end == str)
256                 goto Float;
257         if (value == ULLONG_MAX && errno == ERANGE)
258                 overflow = 1;
```

```
259         while (1) {
260                 char c = *end++;
261                 if (!c) {
262                         break;
263                 } else if (c == 'u' || c == 'U') {
264                         if (want_unsigned)
265                                 goto Enoint;
266                         want_unsigned = 1;
267                 } else if (c == 'l' || c == 'L') {
268                         if (size)
269                                 goto Enoint;
270                         size = 1;
271                         if (*end == c) {
272                                 size = 2;
273                                 end++;
274                         }
275                 } else
276                         goto Float;
277         }
278         if (overflow)
279                 goto Eoverflow;
280         /* OK, it's a valid integer */
281         /* decimals can be unsigned only if directly specified as such */
282         if (str[0] != '0' && !want_unsigned)
283                 try_unsigned = 0;
284         if (!size) {
285                 bits = bits_in_int - 1;
286                 if (!(value & (~1ULL << bits))) {
287                         if (!(value & (1ULL << bits))) {
288                                 goto got_it;
289                         } else if (try_unsigned) {
290                                 want_unsigned = 1;
291                                 goto got_it;
292                         }
293                 }
294                 size = 1;
295                 do_warn = 1;
296         }
297         if (size < 2) {
298                 bits = bits_in_long - 1;
299                 if (!(value & (~1ULL << bits))) {
300                         if (!(value & (1ULL << bits))) {
301                                 goto got_it;
302                         } else if (try_unsigned) {
303                                 want_unsigned = 1;
304                                 goto got_it;
305                         }
306                         do_warn |= 2;
307                 }
308                 size = 2;
309                 do_warn |= 1;
310         }
311         bits = bits_in_longlong - 1;
312         if (value & (~1ULL << bits))
313                 goto Eoverflow;
314         if (!(value & (1ULL << bits)))
315                 goto got_it;
316         if (!try_unsigned)
317                 warning(expr->pos, "decimal constant %s is too big for long long
318                         show_token(token));
319         want_unsigned = 1;
320 got_it:
321         if (do_warn && Wconstant_suffix)
322                 warning(expr->pos, "constant %s is so big it is%s%s%s",
323                         show_token(token),
324                         want_unsigned ? " unsigned":"",
```



```
325                         size > 0 ? " long":"",
326                         size > 1 ? " long":"");
327         if (do_warn & 2)
328                 warning(expr->pos,
329                         "decimal constant %s is between LONG_MAX and ULONG_MAX."
330                         " For C99 that means long long, C90 compilers are very "
331                         "likely to produce unsigned long (and a warning) here",
332                         show_token(token));
333         expr->type = EXPR_VALUE;
334         expr->flags = CEF_SET_INT;
335         expr->ctype = ctype_integer(size, want_unsigned);
336         expr->value = value;
337         return;
338 Eoverflow:
339         error_die(expr->pos, "constant %s is too big even for unsigned long long
340                         show_token(token));
341         return;
342 Float:
343         expr->fvalue = string_to_ld(str, &end);
344         if (str == end)
345                 goto Enoint;

347         if (*end && end[1])
348                 goto Enoint;

350         if (*end == 'f' || *end == 'F')
351                 expr->ctype = &float_ctype;
352         else if (*end == 'l' || *end == 'L')
353                 expr->ctype = &ldouble_ctype;
354         else if (!*end)
355                 expr->ctype = &double_ctype;
356         else
357                 goto Enoint;

359         expr->flags = CEF_SET_FLOAT;
360         expr->type = EXPR_FVALUE;
361         return;

363 Enoint:
364         error_die(expr->pos, "constant %s is not a valid number", show_token(tok
365 }

367 struct token *primary_expression(struct token *token, struct expression **tree)
368 {
369         struct expression *expr = NULL;

371         switch (token_type(token)) {
372         case TOKEN_CHAR ... TOKEN_WIDE_CHAR_EMBEDDED_3:
373                 expr = alloc_expression(token->pos, EXPR_VALUE);
374                 expr->flags = CEF_SET_CHAR;
375                 expr->ctype = token_type(token) < TOKEN_WIDE_CHAR ? &int_ctype :
376                 get_char_constant(token, &expr->value);
377                 token = token->next;
378                 break;

380         case TOKEN_NUMBER:
381                 expr = alloc_expression(token->pos, EXPR_VALUE);
382                 get_number_value(expr, token); /* will see if it's an integer */
383                 token = token->next;
384                 break;

386         case TOKEN_ZERO_IDENT: {
387                 expr = alloc_expression(token->pos, EXPR_SYMBOL);
388                 expr->flags = CEF_SET_INT;
389                 expr->ctype = &int_ctype;
390                 expr->symbol = &zero_int;
```

```
391                 expr->symbol_name = token->ident;
392                 token = token->next;
393                 break;
394         }

396         case TOKEN_IDENT: {
397                 struct symbol *sym = lookup_symbol(token->ident, NS_SYMBOL | NS_
398                 struct token *next = token->next;

400                 if (!sym) {
401                         sym = handle_func(token);
402                         if (token->ident == &__builtin_types_compatible_p_ident)
403                                 token = builtin_types_compatible_p_expr(token, &
404                                 break;
405                         }
406                         if (token->ident == &__builtin_offsetof_ident) {
407                                 token = builtin_offsetof_expr(token, &expr);
408                                 break;
409                         }
410                 } else if (sym->enum_member) {
411                         expr = alloc_expression(token->pos, EXPR_VALUE);
412                         *expr = *sym->initializer;
413                         /* we want the right position reported, thus the copy */
414                         expr->pos = token->pos;
415                         expr->flags = CEF_SET_ENUM;
416                         token = next;
417                         break;
418                 }

420                 expr = alloc_expression(token->pos, EXPR_SYMBOL);

422                 /*
423                  * We support types as real first-class citizens, with type
424                  * comparisons etc:
425                  *
426                  *      if (typeof(a) == int) ..
427                  */
428                 if (sym && sym->namespace == NS_TYPEDEF) {
429                         sparse_error(token->pos, "typename in expression");
430                         sym = NULL;
431                 }
432                 expr->symbol_name = token->ident;
433                 expr->symbol = sym;

435                 /*
436                  * A pointer to an lvalue designating a static storage
437                  * duration object is an address constant [6.6(9)].
438                  */
439                 if (sym && (sym->ctype.modifiers & (MOD_TOPLEVEL | MOD_STATIC)))
440                         expr->flags = CEF_ADDR;

442                 token = next;
443                 break;
444         }

446         case TOKEN_STRING:
447         case TOKEN_WIDE_STRING:
448                 expr = alloc_expression(token->pos, EXPR_STRING);
449                 token = get_string_constant(token, expr);
450                 break;

452         case TOKEN_SPECIAL:
453                 if (token->special == '(') {
454                         expr = alloc_expression(token->pos, EXPR_PREOP);
455                         expr->op = '(';
456                         token = parens_expression(token, &expr->unop, "in expres
```



```
457                         break;
458                 }
459                 if (token->special == '[' && lookup_type(token->next)) {
460                         expr = alloc_expression(token->pos, EXPR_TYPE);
461                         token = typename(token->next, &expr->symbol, NULL);
462                         token = expect(token, ']', "in type expression");
463                         break;
464                 }
465
466         default:
467                 ;
468         }
469         *tree = expr;
470         return token;
471 }

473 static struct token *expression_list(struct token *token, struct expression_list
474 {
475         while (!match_op(token, ')')) {
476                 struct expression *expr = NULL;
477                 token = assignment_expression(token, &expr);
478                 if (!expr)
479                         break;
480                 add_expression(list, expr);
481                 if (!match_op(token, ','))
482                         break;
483                 token = token->next;
484         }
485         return token;
486 }

488 /*
489  * extend to deal with the ambiguous C grammar for parsing
490  * a cast expressions followed by an initializer.
491  */
492 static struct token *postfix_expression(struct token *token, struct expression *
493 {
494         struct expression *expr = cast_init_expr;

496         if (!expr)
497                 token = primary_expression(token, &expr);

499         while (expr && token_type(token) == TOKEN_SPECIAL) {
500                 switch (token->special) {
501                 case '[': {                     /* Array dereference */
502                         struct expression *deref = alloc_expression(token->pos,
503                         struct expression *add = alloc_expression(token->pos, EX

505                         deref->op = '*';
506                         deref->unop = add;

508                         add->op = '+';
509                         add->left = expr;
510                         token = parse_expression(token->next, &add->right);
511                         token = expect(token, ']', "at end of array dereference"
512                         expr = deref;
513                         continue;
514                 }
515                 case SPECIAL_INCREMENT:         /* Post-increment */
516                 case SPECIAL_DECREMENT: {       /* Post-decrement */
517                         struct expression *post = alloc_expression(token->pos, E
518                         post->op = token->special;
519                         post->unop = expr;
520                         expr = post;
521                         token = token->next;
522                         continue;
```

```
523                 }
524                 case SPECIAL_DEREFERENCE: {     /* Structure pointer member dere
525                         /* "x->y" is just shorthand for "(*x).y" */
526                         struct expression *inner = alloc_expression(token->pos,
527                         inner->op = '*';
528                         inner->unop = expr;
529                         expr = inner;
530                 }
531                 /* Fall through!! */
532                 case '.': {                     /* Structure member dereference
533                         struct expression *deref = alloc_expression(token->pos,
534                         deref->op = '.';
535                         deref->deref = expr;
536                         token = token->next;
537                         if (token_type(token) != TOKEN_IDENT) {
538                                 sparse_error(token->pos, "Expected member name")
539                                 break;
540                         }
541                         deref->member = token->ident;
542                         deref->member_offset = -1;
543                         token = token->next;
544                         expr = deref;
545                         continue;
546                 }

548                 case '(': {                     /* Function call */
549                         struct expression *call = alloc_expression(token->pos, E
550                         call->op = '(';
551                         call->fn = expr;
552                         token = expression_list(token->next, &call->args);
553                         token = expect(token, ')', "in function call");
554                         expr = call;
555                         continue;
556                 }

558                 default:
559                         break;
560                 }
561                 break;
562         }
563         *tree = expr;
564         return token;
565 }

567 static struct token *cast_expression(struct token *token, struct expression **tr
568 static struct token *unary_expression(struct token *token, struct expression **t

570 static struct token *type_info_expression(struct token *token,
571         struct expression **tree, int type)
572 {
573         struct expression *expr = alloc_expression(token->pos, type);
574         struct token *p;

576         *tree = expr;
577         expr->flags = CEF_SET_ICE; /* XXX: VLA support will need that changed */
578         token = token->next;
579         if (!match_op(token, '(') || !lookup_type(token->next))
580                 return unary_expression(token, &expr->cast_expression);
581         p = token;
582         token = typename(token->next, &expr->cast_type, NULL);

584         if (!match_op(token, ')')) {
585                 static const char * error[] = {
586                         [EXPR_SIZEOF] = "at end of sizeof",
587                         [EXPR_ALIGNOF] = "at end of __alignof__",
588                         [EXPR_PTRSIZEOF] = "at end of __sizeof_ptr__"
```



```
589                 };
590                 return expect(token, ')', error[type]);
591         }

593         token = token->next;
594         /*
595          * C99 ambiguity: the typename might have been the beginning
596          * of a typed initializer expression..
597          */
598         if (match_op(token, '{')) {
599                 struct expression *cast = alloc_expression(p->pos, EXPR_CAST);
600                 cast->cast_type = expr->cast_type;
601                 expr->cast_type = NULL;
602                 expr->cast_expression = cast;
603                 token = initializer(&cast->cast_expression, token);
604                 token = postfix_expression(token, &expr->cast_expression, cast);
605         }
606         return token;
607 }

609 static struct token *unary_expression(struct token *token, struct expression **t
610 {
611         if (token_type(token) == TOKEN_IDENT) {
612                 struct ident *ident = token->ident;
613                 if (ident->reserved) {
614                         static const struct {
615                                 struct ident *id;
616                                 int type;
617                         } type_information[] = {
618                                 { &sizeof_ident, EXPR_SIZEOF },
619                                 { &__alignof___ident, EXPR_ALIGNOF },
620                                 { &__alignof_ident, EXPR_ALIGNOF },
621                                 { &_Alignof_ident, EXPR_ALIGNOF },
622                                 { &__sizeof_ptr___ident, EXPR_PTRSIZEOF },
623                         };
624                         int i;
625                         for (i = 0; i < ARRAY_SIZE(type_information); i++) {
626                                 if (ident == type_information[i].id)
627                                         return type_info_expression(token, tree,
628                         }
629                 }
630         }

632         if (token_type(token) == TOKEN_SPECIAL) {
633                 if (match_oplist(token->special,
634                     SPECIAL_INCREMENT, SPECIAL_DECREMENT,
635                     '&', '*', 0)) {
636                         struct expression *unop;
637                         struct expression *unary;
638                         struct token *next;

640                         next = cast_expression(token->next, &unop);
641                         if (!unop) {
642                                 sparse_error(token->pos, "Syntax error in unary
643                                 *tree = NULL;
644                                 return next;
645                         }
646                         unary = alloc_expression(token->pos, EXPR_PREOP);
647                         unary->op = token->special;
648                         unary->unop = unop;
649                         *tree = unary;
650                         return next;
651                 }
652                 /* possibly constant ones */
653                 if (match_oplist(token->special, '+', '-', '~', '!', 0)) {
654                         struct expression *unop;
```

```
655                         struct expression *unary;
656                         struct token *next;

658                         next = cast_expression(token->next, &unop);
659                         if (!unop) {
660                                 sparse_error(token->pos, "Syntax error in unary
661                                 *tree = NULL;
662                                 return next;
663                         }
664                         unary = alloc_expression(token->pos, EXPR_PREOP);
665                         unary->op = token->special;
666                         unary->unop = unop;
667                         *tree = unary;
668                         return next;
669                 }
670                 /* Gcc extension: &&label gives the address of a label */
671                 if (match_op(token, SPECIAL_LOGICAL_AND) &&
672                     token_type(token->next) == TOKEN_IDENT) {
673                         struct expression *label = alloc_expression(token->pos,
674                         struct symbol *sym = label_symbol(token->next);
675                         if (!(sym->ctype.modifiers & MOD_ADDRESSABLE)) {
676                                 sym->ctype.modifiers |= MOD_ADDRESSABLE;
677                                 add_symbol(&function_computed_target_list, sym);
678                         }
679                         label->flags = CEF_ADDR;
680                         label->label_symbol = sym;
681                         *tree = label;
682                         return token->next->next;
683                 }
684
685         }
686
687         return postfix_expression(token, tree, NULL);
688 }

690 /*
691  * Ambiguity: a '(' can be either a cast-expression or
692  * a primary-expression depending on whether it is followed
693  * by a type or not.
694  *
695  * additional ambiguity: a "cast expression" followed by
696  * an initializer is really a postfix-expression.
697  */
698 static struct token *cast_expression(struct token *token, struct expression **tr
699 {
700         if (match_op(token, '(')) {
701                 struct token *next = token->next;
702                 if (lookup_type(next)) {
703                         struct expression *cast = alloc_expression(next->pos, EX
704                         struct expression *v;
705                         struct symbol *sym;
706                         int is_force;

708                         token = typename(next, &sym, &is_force);
709                         cast->cast_type = sym;
710                         token = expect(token, ')', "at end of cast operator");
711                         if (match_op(token, '{')) {
712                                 if (toplevel(block_scope))
713                                         sym->ctype.modifiers |= MOD_TOPLEVEL;
714                                 if (is_force)
715                                         warning(sym->pos,
716                                                 "[force] in compound literal");
717                                 token = initializer(&cast->cast_expression, toke
718                                 return postfix_expression(token, tree, cast);
719                         }
720                         *tree = cast;
```



```
721                         if (is_force)
722                                 cast->type = EXPR_FORCE_CAST;
723                         token = cast_expression(token, &v);
724                         if (!v)
725                                 return token;
726                         cast->cast_expression = v;
727                         return token;
728                 }
729         }
730         return unary_expression(token, tree);
731 }

733 /*
734  * Generic left-to-right binop parsing
735  *
736  * This _really_ needs to be inlined, because that makes the inner
737  * function call statically deterministic rather than a totally
738  * unpredictable indirect call. But gcc-3 is so "clever" that it
739  * doesn't do so by default even when you tell it to inline it.
740  *
741  * Making it a macro avoids the inlining problem, and also means
742  * that we can pass in the op-comparison as an expression rather
743  * than create a data structure for it.
744  */

746 #define LR_BINOP_EXPRESSION(__token, tree, type, inner, compare)        \
747         struct expression *left = NULL;                                 \
748         struct token * next = inner(__token, &left);                    \
749                                                                         \
750         if (left) {                                                     \
751                 while (token_type(next) == TOKEN_SPECIAL) {             \
752                         struct expression *top, *right = NULL;          \
753                         int op = next->special;                         \
754                                                                         \
755                         if (!(compare))                                 \
756                                 goto out;                               \
757                         top = alloc_expression(next->pos, type);        \
758                         next = inner(next->next, &right);               \
759                         if (!right) {                                   \
760                                 sparse_error(next->pos, "No right hand side of '
761                                 break;                                  \
762                         }                                               \
763                         top->op = op;                                   \
764                         top->left = left;                               \
765                         top->right = right;                             \
766                         left = top;                                     \
767                 }                                                       \
768         }                                                               \
769 out:                                                                    \
770         *tree = left;                                                   \
771         return next;                                                    \

773 static struct token *multiplicative_expression(struct token *token, struct expre
774 {
775         LR_BINOP_EXPRESSION(
776                 token, tree, EXPR_BINOP, cast_expression,
777                 (op == '*') || (op == '/') || (op == '%')
778         );
779 }

781 static struct token *additive_expression(struct token *token, struct expression
782 {
783         LR_BINOP_EXPRESSION(
784                 token, tree, EXPR_BINOP, multiplicative_expression,
785                 (op == '+') || (op == '-')
786         );
```

```
}

static struct token *shift_expression(struct token *token, struct expression **t
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_BINOP, additive_expression,
                (op == SPECIAL_LEFTSHIFT) || (op == SPECIAL_RIGHTSHIFT)
        );
}

static struct token *relational_expression(struct token *token, struct expressio
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_COMPARE, shift_expression,
                (op == '<') || (op == '>') ||
                (op == SPECIAL_LTE) || (op == SPECIAL_GTE)
        );
}

static struct token *equality_expression(struct token *token, struct expression
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_COMPARE, relational_expression,
                (op == SPECIAL_EQUAL) || (op == SPECIAL_NOTEQUAL)
        );
}

static struct token *bitwise_and_expression(struct token *token, struct expressi
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_BINOP, equality_expression,
                (op == '&')
        );
}

static struct token *bitwise_xor_expression(struct token *token, struct expressi
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_BINOP, bitwise_and_expression,
                (op == '^')
        );
}

static struct token *bitwise_or_expression(struct token *token, struct expressio
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_BINOP, bitwise_xor_expression,
                (op == '|')
        );
}

static struct token *logical_and_expression(struct token *token, struct expressi
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_LOGICAL, bitwise_or_expression,
                (op == SPECIAL_LOGICAL_AND)
        );
}

static struct token *logical_or_expression(struct token *token, struct expressio
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_LOGICAL, logical_and_expression,
                (op == SPECIAL_LOGICAL_OR)
        );
}
```



```
struct token *conditional_expression(struct token *token, struct expression **tr
{
        token = logical_or_expression(token, tree);
        if (*tree && match_op(token, '?')) {
                struct expression *expr = alloc_expression(token->pos, EXPR_COND
                expr->op = token->special;
                expr->left = *tree;
                *tree = expr;
                token = parse_expression(token->next, &expr->cond_true);
                token = expect(token, ':', "in conditional expression");
                token = conditional_expression(token, &expr->cond_false);
        }
        return token;
}

struct token *assignment_expression(struct token *token, struct expression **tre
{
        token = conditional_expression(token, tree);
        if (*tree && token_type(token) == TOKEN_SPECIAL) {
                static const int assignments[] = {
                        '=',
                        SPECIAL_ADD_ASSIGN, SPECIAL_SUB_ASSIGN,
                        SPECIAL_MUL_ASSIGN, SPECIAL_DIV_ASSIGN,
                        SPECIAL_MOD_ASSIGN, SPECIAL_SHL_ASSIGN,
                        SPECIAL_SHR_ASSIGN, SPECIAL_AND_ASSIGN,
                        SPECIAL_OR_ASSIGN,  SPECIAL_XOR_ASSIGN };
                int i, op = token->special;
                for (i = 0; i < ARRAY_SIZE(assignments); i++)
                        if (assignments[i] == op) {
                                struct expression * expr = alloc_expression(toke
                                expr->left = *tree;
                                expr->op = op;
                                *tree = expr;
                                return assignment_expression(token->next, &expr-
                        }
        }
        return token;
}

static struct token *comma_expression(struct token *token, struct expression **t
{
        LR_BINOP_EXPRESSION(
                token, tree, EXPR_COMMA, assignment_expression,
                (op == ',')
        );
}

struct token *parse_expression(struct token *token, struct expression **tree)
{
        return comma_expression(token,tree);
}
```

```
**********************************************************
    8355 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/expression.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef EXPRESSION_H
#define EXPRESSION_H
/*
 * sparse/expression.h
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * Declarations and helper functions for expression parsing.
 */

#include "allocate.h"
#include "lib.h"
#include "symbol.h"

struct expression_list;

enum expression_type {
        EXPR_VALUE = 1,
        EXPR_STRING,
        EXPR_SYMBOL,
        EXPR_TYPE,
        EXPR_BINOP,
        EXPR_ASSIGNMENT,
        EXPR_LOGICAL,
        EXPR_DEREF,
        EXPR_PREOP,
        EXPR_POSTOP,
        EXPR_CAST,
        EXPR_FORCE_CAST,
        EXPR_IMPLIED_CAST,
        EXPR_SIZEOF,
        EXPR_ALIGNOF,
        EXPR_PTRSIZEOF,
        EXPR_CONDITIONAL,
        EXPR_SELECT,            // a "safe" conditional expression
        EXPR_STATEMENT,
        EXPR_CALL,
        EXPR_COMMA,
        EXPR_COMPARE,
        EXPR_LABEL,
        EXPR_INITIALIZER,       // initializer list
```



```
        EXPR_IDENTIFIER,        // identifier in initializer
        EXPR_INDEX,             // index in initializer
        EXPR_POS,               // position in initializer
        EXPR_FVALUE,
        EXPR_SLICE,
        EXPR_OFFSETOF,
};


/*
 * Flags for tracking the promotion of constness related attributes
 * from subexpressions to their parents.
 *
 * The flags are not independent as one might imply another.
 * The implications are as follows:
 * - CEF_INT, CEF_ENUM and
 *   CEF_CHAR imply CEF_ICE.
 *
 * Use the CEF_*_SET_MASK and CEF_*_CLEAR_MASK
 * helper macros defined below to set or clear one of these flags.
 */
enum constexpr_flag {
        CEF_NONE = 0,
        /*
         * A constant in the sense of [6.4.4]:
         * - Integer constant [6.4.4.1]
         * - Floating point constant [6.4.4.2]
         * - Enumeration constant [6.4.4.3]
         * - Character constant [6.4.4.4]
         */
        CEF_INT = (1 << 0),
        CEF_FLOAT = (1 << 1),
        CEF_ENUM = (1 << 2),
        CEF_CHAR = (1 << 3),

        /*
         * A constant expression in the sense of [6.6]:
         * - integer constant expression [6.6(6)]
         * - arithmetic constant expression [6.6(8)]
         * - address constant [6.6(9)]
         */
        CEF_ICE = (1 << 4),
        CEF_ACE = (1 << 5),
        CEF_ADDR = (1 << 6),

        /* integer constant expression => arithmetic constant expression */
        CEF_SET_ICE = (CEF_ICE | CEF_ACE),

        /* integer constant => integer constant expression */
        CEF_SET_INT = (CEF_INT | CEF_SET_ICE),

        /* floating point constant => arithmetic constant expression */
        CEF_SET_FLOAT = (CEF_FLOAT | CEF_ACE),

        /* enumeration constant => integer constant expression */
        CEF_SET_ENUM = (CEF_ENUM | CEF_SET_ICE),

        /* character constant => integer constant expression */
        CEF_SET_CHAR = (CEF_CHAR | CEF_SET_ICE),

        /*
         * Remove any "Constant" [6.4.4] flag, but retain the "constant
         * expression" [6.6] flags.
         */
        CEF_CONST_MASK = (CEF_INT | CEF_FLOAT | CEF_CHAR),
```

```
127         /*
128          * not an integer constant expression => neither of integer,
129          * enumeration and character constant
130          */
131         CEF_CLR_ICE = (CEF_ICE | CEF_INT | CEF_ENUM | CEF_CHAR),
132 };

133 enum {
134         Handled = 1 << 0,
135         Fake    = 1 << 1,
136 }; /* for expr->flags */

137 enum {
138         Taint_comma = 1,
139 }; /* for expr->taint */

140 struct expression {
141         enum expression_type type:8;
142         unsigned flags:8;
143         unsigned smatch_flags:16;
144         int op;
145         struct position pos;
146         struct symbol *ctype;
147         unsigned long parent;
148         union {
149                 // EXPR_VALUE
150                 struct {
151                         unsigned long long value;
152                         unsigned taint;
153                 };

154                 // EXPR_FVALUE
155                 long double fvalue;

156                 // EXPR_STRING
157                 struct {
158                         int wide;
159                         struct string *string;
160                 };

161                 // EXPR_UNOP, EXPR_PREOP and EXPR_POSTOP
162                 struct /* unop */ {
163                         struct expression *unop;
164                         unsigned long op_value;
165                 };

166                 // EXPR_SYMBOL, EXPR_TYPE
167                 struct /* symbol_arg */ {
168                         struct symbol *symbol;
169                         struct ident *symbol_name;
170                 };

171                 // EXPR_STATEMENT
172                 struct statement *statement;

173                 // EXPR_BINOP, EXPR_COMMA, EXPR_COMPARE, EXPR_LOGICAL and EXPR_A
174                 struct /* binop_arg */ {
175                         struct expression *left, *right;
176                 };
177                 // EXPR_DEREF
178                 struct /* deref_arg */ {
179                         struct expression *deref;
180                         struct ident *member;
181                         int member_offset;
182                 };
183                 // EXPR_SLICE
```



```
184                 struct /* slice */ {
185                         struct expression *base;
186                         unsigned r_bitpos, r_nrbits;
187                 };
188                 // EXPR_CAST and EXPR_SIZEOF
189                 struct /* cast_arg */ {
190                         struct symbol *cast_type;
191                         struct expression *cast_expression;
192                 };
193                 // EXPR_CONDITIONAL
194                 // EXPR_SELECT
195                 struct /* conditional_expr */ {
196                         struct expression *conditional, *cond_true, *cond_false;
197                 };
198                 // EXPR_CALL
199                 struct /* call_expr */ {
200                         struct expression *fn;
201                         struct expression_list *args;
202                 };
203                 // EXPR_LABEL
204                 struct /* label_expr */ {
205                         struct symbol *label_symbol;
206                 };
207                 // EXPR_INITIALIZER
208                 struct expression_list *expr_list;
209                 // EXPR_IDENTIFIER
210                 struct /* ident_expr */ {
211                         int offset;
212                         struct ident *expr_ident;
213                         struct symbol *field;
214                         struct expression *ident_expression;
215                 };
216                 // EXPR_INDEX
217                 struct /* index_expr */ {
218                         unsigned int idx_from, idx_to;
219                         struct expression *idx_expression;
220                 };
221                 // EXPR_POS
222                 struct /* initpos_expr */ {
223                         unsigned int init_offset, init_nr;
224                         struct expression *init_expr;
225                 };
226                 // EXPR_OFFSETOF
227                 struct {
228                         struct symbol *in;
229                         struct expression *down;
230                         union {
231                                 struct ident *ident;
232                                 struct expression *index;
233                         };
234                 };
235         };
236 };

237 /* Constant expression values */
238 int is_zero_constant(struct expression *);
239 int expr_truth_value(struct expression *expr);
240 long long get_expression_value(struct expression *);
241 long long const_expression_value(struct expression *);
242 long long get_expression_value_silent(struct expression *expr);

243 /* Expression parsing */
244 struct token *parse_expression(struct token *token, struct expression **tree);
245 struct token *conditional_expression(struct token *token, struct expression **tr
246 struct token *primary_expression(struct token *token, struct expression **tree);
247 struct token *parens_expression(struct token *token, struct expression **expr, c
```

```
259 struct token *assignment_expression(struct token *token, struct expression **tre

261 extern void evaluate_symbol_list(struct symbol_list *list);
262 extern struct symbol *evaluate_statement(struct statement *stmt);
263 extern struct symbol *evaluate_expression(struct expression *);

265 extern int expand_symbol(struct symbol *);

267 static inline struct expression *alloc_expression(struct position pos, int type)
268 {
269         struct expression *expr = __alloc_expression(0);
270         expr->type = type;
271         expr->pos = pos;
272         expr->flags = CEF_NONE;
273         return expr;
274 }

276 static inline struct expression *alloc_const_expression(struct position pos, int
277 {
278         struct expression *expr = __alloc_expression(0);
279         expr->type = EXPR_VALUE;
280         expr->pos = pos;
281         expr->value = value;
282         expr->ctype = &int_ctype;
283         expr->flags = CEF_SET_INT;
284         return expr;
285 }

287 /* Type name parsing */
288 struct token *typename(struct token *, struct symbol **, int *);

290 static inline int lookup_type(struct token *token)
291 {
292         if (token->pos.type == TOKEN_IDENT) {
293                 struct symbol *sym = lookup_symbol(token->ident, NS_SYMBOL | NS_
294                 return sym && (sym->namespace & NS_TYPEDEF);
295         }
296         return 0;
297 }

299 /* Statement parsing */
300 struct statement *alloc_statement(struct position pos, int type);
301 struct token *initializer(struct expression **tree, struct token *token);
302 struct token *compound_statement(struct token *, struct statement *);

304 /* The preprocessor calls this 'constant_expression()' */
305 #define constant_expression(token,tree) conditional_expression(token, tree)

307 /* Cast folding of constant values.. */
308 void cast_value(struct expression *expr, struct symbol *newtype,
309         struct expression *old, struct symbol *oldtype);

311 #endif
```

```
**********************************************************
   23959 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/flow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Flow - walk the linearized flowgraph, simplifying it as we
 * go along.
 *
 * Copyright (C) 2004 Linus Torvalds
 */

#include <string.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <assert.h>

#include "parse.h"
#include "expression.h"
#include "linearize.h"
#include "flow.h"
#include "target.h"

unsigned long bb_generation;

/*
 * Dammit, if we have a phi-node followed by a conditional
 * branch on that phi-node, we should damn well be able to
 * do something about the source. Maybe.
 */
static int rewrite_branch(struct basic_block *bb,
        struct basic_block **ptr,
        struct basic_block *old,
        struct basic_block *new)
{
        if (*ptr != old || new == old || !bb->ep)
                return 0;

        /* We might find new if-conversions or non-dominating CSEs */
        /* we may also create new dead cycles */
        repeat_phase |= REPEAT_CSE | REPEAT_CFG_CLEANUP;
        *ptr = new;
        replace_bb_in_list(&bb->children, old, new, 1);
        remove_bb_from_list(&old->parents, bb, 1);
        add_bb(&new->parents, bb);
        return 1;
}

/*
 * Return the known truth value of a pseudo, or -1 if
 * it's not known.
 */
static int pseudo_truth_value(pseudo_t pseudo)
{
        switch (pseudo->type) {
        case PSEUDO_VAL:
                return !!pseudo->value;

        case PSEUDO_REG: {
                struct instruction *insn = pseudo->def;

                /* A symbol address is always considered true.. */
                if (insn->opcode == OP_SYMADDR && insn->target == pseudo)
```



```
                        return 1;
        }
                /* Fall through */
        default:
                return -1;
        }
}

/*
 * Does a basic block depend on the pseudos that "src" defines?
 */
static int bb_depends_on(struct basic_block *target, struct basic_block *src)
{
        pseudo_t pseudo;

        FOR_EACH_PTR(src->defines, pseudo) {
                if (pseudo_in_list(target->needs, pseudo))
                        return 1;
        } END_FOR_EACH_PTR(pseudo);
        return 0;
}

/*
 * This really should be handled by bb_depends_on()
 * which efficiently check the dependence using the
 * defines - needs liveness info. Problem is that
 * there is no liveness done on OP_PHI & OP_PHISRC.
 *
 * This function add the missing dependency checks.
 */
static int bb_depends_on_phi(struct basic_block *target, struct basic_block *src
{
        struct instruction *insn;
        FOR_EACH_PTR(src->insns, insn) {
                if (!insn->bb)
                        continue;
                if (insn->opcode != OP_PHI)
                        continue;
                if (pseudo_in_list(target->needs, insn->target))
                        return 1;
        } END_FOR_EACH_PTR(insn);
        return 0;
}

/*
 * When we reach here, we have:
 *  - a basic block that ends in a conditional branch and
 *    that has no side effects apart from the pseudos it
 *    may change.
 *  - the phi-node that the conditional branch depends on
 *  - full pseudo liveness information
 *
 * We need to check if any of the _sources_ of the phi-node
 * may be constant, and not actually need this block at all.
 */
static int try_to_simplify_bb(struct basic_block *bb, struct instruction *first,
{
        int changed = 0;
        pseudo_t phi;
        int bogus;

        /*
         * This a due to improper dominance tracking during
         * simplify_symbol_usage()/conversion to SSA form.
         * No sane simplification can be done when we have this.
         */
```

```
127         bogus = bb_list_size(bb->parents) != pseudo_list_size(first->phi_list);

129         FOR_EACH_PTR(first->phi_list, phi) {
130                 struct instruction *def = phi->def;
131                 struct basic_block *source, *target;
132                 pseudo_t pseudo;
133                 struct instruction *br;
134                 int true;

136                 if (!def)
137                         continue;
138                 source = def->bb;
139                 pseudo = def->src1;
140                 if (!pseudo || !source)
141                         continue;
142                 br = last_instruction(source->insns);
143                 if (!br)
144                         continue;
145                 if (br->opcode != OP_CBR && br->opcode != OP_BR)
146                         continue;
147                 true = pseudo_truth_value(pseudo);
148                 if (true < 0)
149                         continue;
150                 target = true ? second->bb_true : second->bb_false;
151                 if (bb_depends_on(target, bb))
152                         continue;
153                 if (bb_depends_on_phi(target, bb))
154                         continue;
155                 changed |= rewrite_branch(source, &br->bb_true, bb, target);
156                 changed |= rewrite_branch(source, &br->bb_false, bb, target);
157                 if (changed && !bogus)
158                         kill_use(THIS_ADDRESS(phi));
159         } END_FOR_EACH_PTR(phi);
160         return changed;
161 }

163 static int bb_has_side_effects(struct basic_block *bb)
164 {
165         struct instruction *insn;
166         FOR_EACH_PTR(bb->insns, insn) {
167                 switch (insn->opcode) {
168                 case OP_CALL:
169                         /* FIXME! This should take "const" etc into account */
170                         return 1;

172                 case OP_STORE:
173                 case OP_CONTEXT:
174                         return 1;

176                 case OP_ASM:
177                         /* FIXME! This should take "volatile" etc into account *
178                         return 1;

180                 default:
181                         continue;
182                 }
183         } END_FOR_EACH_PTR(insn);
184         return 0;
185 }

187 static int simplify_phi_branch(struct basic_block *bb, struct instruction *br)
188 {
189         pseudo_t cond = br->cond;
190         struct instruction *def;

192         if (cond->type != PSEUDO_REG)
```



```
193                 return 0;
194         def = cond->def;
195         if (def->bb != bb || def->opcode != OP_PHI)
196                 return 0;
197         if (bb_has_side_effects(bb))
198                 return 0;
199         return try_to_simplify_bb(bb, def, br);
200 }

202 static int simplify_branch_branch(struct basic_block *bb, struct instruction *br
203         struct basic_block **target_p, int true)
204 {
205         struct basic_block *target = *target_p, *final;
206         struct instruction *insn;
207         int retval;

209         if (target == bb)
210                 return 0;
211         insn = last_instruction(target->insns);
212         if (!insn || insn->opcode != OP_CBR || insn->cond != br->cond)
213                 return 0;
214         /*
215          * Ahhah! We've found a branch to a branch on the same conditional!
216          * Now we just need to see if we can rewrite the branch..
217          */
218         retval = 0;
219         final = true ? insn->bb_true : insn->bb_false;
220         if (bb_has_side_effects(target))
221                 goto try_to_rewrite_target;
222         if (bb_depends_on(final, target))
223                 goto try_to_rewrite_target;
224         if (bb_depends_on_phi(final, target))
225                 return 0;
226         return rewrite_branch(bb, target_p, target, final);

228 try_to_rewrite_target:
229         /*
230          * If we're the only parent, at least we can rewrite the
231          * now-known second branch.
232          */
233         if (bb_list_size(target->parents) != 1)
234                 return retval;
235         insert_branch(target, insn, final);
236         return 1;
237 }

239 static int simplify_one_branch(struct basic_block *bb, struct instruction *br)
240 {
241         if (simplify_phi_branch(bb, br))
242                 return 1;
243         return simplify_branch_branch(bb, br, &br->bb_true, 1) |
244                simplify_branch_branch(bb, br, &br->bb_false, 0);
245 }

247 static int simplify_branch_nodes(struct entrypoint *ep)
248 {
249         int changed = 0;
250         struct basic_block *bb;

252         FOR_EACH_PTR(ep->bbs, bb) {
253                 struct instruction *br = last_instruction(bb->insns);

255                 if (!br || br->opcode != OP_CBR)
256                         continue;
257                 changed |= simplify_one_branch(bb, br);
258         } END_FOR_EACH_PTR(bb);
```

```
259         return changed;
260 }

262 /*
263  * This is called late - when we have intra-bb liveness information..
264  */
265 int simplify_flow(struct entrypoint *ep)
266 {
267         return simplify_branch_nodes(ep);
268 }

270 static inline void concat_user_list(struct pseudo_user_list *src, struct pseudo_
271 {
272         concat_ptr_list((struct ptr_list *)src, (struct ptr_list **)dst);
273 }

275 void convert_instruction_target(struct instruction *insn, pseudo_t src)
276 {
277         pseudo_t target;
278         struct pseudo_user *pu;
279         /*
280          * Go through the "insn->users" list and replace them all..
281          */
282         target = insn->target;
283         if (target == src)
284                 return;
285         FOR_EACH_PTR(target->users, pu) {
286                 if (*pu->userp != VOID) {
287                         assert(*pu->userp == target);
288                         *pu->userp = src;
289                 }
290         } END_FOR_EACH_PTR(pu);
291         if (has_use_list(src))
292                 concat_user_list(target->users, &src->users);
293         target->users = NULL;
294 }

296 void convert_load_instruction(struct instruction *insn, pseudo_t src)
297 {
298         convert_instruction_target(insn, src);
299         /* Turn the load into a no-op */
300         insn->opcode = OP_LNOP;
301         insn->bb = NULL;
302 }

304 static int overlapping_memop(struct instruction *a, struct instruction *b)
305 {
306         unsigned int a_start = bytes_to_bits(a->offset);
307         unsigned int b_start = bytes_to_bits(b->offset);
308         unsigned int a_size = a->size;
309         unsigned int b_size = b->size;

311         if (a_size + a_start <= b_start)
312                 return 0;
313         if (b_size + b_start <= a_start)
314                 return 0;
315         return 1;
316 }

318 static inline int same_memop(struct instruction *a, struct instruction *b)
319 {
320         return  a->offset == b->offset && a->size == b->size;
321 }

323 static inline int distinct_symbols(pseudo_t a, pseudo_t b)
324 {
```



```
325         if (a->type != PSEUDO_SYM)
326                 return 0;
327         if (b->type != PSEUDO_SYM)
328                 return 0;
329         return a->sym != b->sym;
330 }

332 /*
333  * Return 1 if "dom" dominates the access to "pseudo"
334  * in "insn".
335  *
336  * Return 0 if it doesn't, and -1 if you don't know.
337  */
338 int dominates(pseudo_t pseudo, struct instruction *insn, struct instruction *dom
339 {
340         int opcode = dom->opcode;

342         if (opcode == OP_CALL || opcode == OP_ENTRY)
343                 return local ? 0 : -1;
344         if (opcode != OP_LOAD && opcode != OP_STORE)
345                 return 0;
346         if (dom->src != pseudo) {
347                 if (local)
348                         return 0;
349                 /* We don't think two explicitly different symbols ever alias */
350                 if (distinct_symbols(insn->src, dom->src))
351                         return 0;
352                 /* We could try to do some alias analysis here */
353                 return -1;
354         }
355         if (!same_memop(insn, dom)) {
356                 if (dom->opcode == OP_LOAD)
357                         return 0;
358                 if (!overlapping_memop(insn, dom))
359                         return 0;
360                 return -1;
361         }
362         return 1;
363 }

365 static int phisrc_in_bb(struct pseudo_list *list, struct basic_block *bb)
366 {
367         pseudo_t p;
368         FOR_EACH_PTR(list, p) {
369                 if (p->def->bb == bb)
370                         return 1;
371         } END_FOR_EACH_PTR(p);

373         return 0;
374 }

376 static int find_dominating_parents(pseudo_t pseudo, struct instruction *insn,
377         struct basic_block *bb, unsigned long generation, struct pseudo_list **d
378         int local)
379 {
380         struct basic_block *parent;

382         if (!bb->parents)
383                 return !!local;

385         FOR_EACH_PTR(bb->parents, parent) {
386                 struct instruction *one;
387                 struct instruction *br;
388                 pseudo_t phi;

390                 FOR_EACH_PTR_REVERSE(parent->insns, one) {
```

```
391                         int dominance;
392                         if (one == insn)
393                                 goto no_dominance;
394                         dominance = dominates(pseudo, insn, one, local);
395                         if (dominance < 0) {
396                                 if (one->opcode == OP_LOAD)
397                                         continue;
398                                 return 0;
399                         }
400                         if (!dominance)
401                                 continue;
402                         goto found_dominator;
403                 } END_FOR_EACH_PTR_REVERSE(one);
404 no_dominance:
405                 if (parent->generation == generation)
406                         continue;
407                 parent->generation = generation;

409                 if (!find_dominating_parents(pseudo, insn, parent, generation, d
410                         return 0;
411                 continue;

413 found_dominator:
414                 if (dominators && phisrc_in_bb(*dominators, parent))
415                         continue;
416                 br = delete_last_instruction(&parent->insns);
417                 phi = alloc_phi(parent, one->target, one->size);
418                 phi->ident = phi->ident ? : pseudo->ident;
419                 add_instruction(&parent->insns, br);
420                 use_pseudo(insn, phi, add_pseudo(dominators, phi));
421         } END_FOR_EACH_PTR(parent);
422         return 1;
423 }

425 /*
426  * We should probably sort the phi list just to make it easier to compare
427  * later for equality.
428  */
429 void rewrite_load_instruction(struct instruction *insn, struct pseudo_list *domi
430 {
431         pseudo_t new, phi;

433         /*
434          * Check for somewhat common case of duplicate
435          * phi nodes.
436          */
437         new = first_pseudo(dominators)->def->src1;
438         FOR_EACH_PTR(dominators, phi) {
439                 if (new != phi->def->src1)
440                         goto complex_phi;
441                 new->ident = new->ident ? : phi->ident;
442         } END_FOR_EACH_PTR(phi);

444         /*
445          * All the same pseudo - mark the phi-nodes unused
446          * and convert the load into a LNOP and replace the
447          * pseudo.
448          */
449         FOR_EACH_PTR(dominators, phi) {
450                 kill_instruction(phi->def);
451         } END_FOR_EACH_PTR(phi);
452         convert_load_instruction(insn, new);
453         return;

455 complex_phi:
456         /* We leave symbol pseudos with a bogus usage list here */
```



```
457         if (insn->src->type != PSEUDO_SYM)
458                 kill_use(&insn->src);
459         insn->opcode = OP_PHI;
460         insn->phi_list = dominators;
461 }

463 static int find_dominating_stores(pseudo_t pseudo, struct instruction *insn,
464         unsigned long generation, int local)
465 {
466         struct basic_block *bb = insn->bb;
467         struct instruction *one, *dom = NULL;
468         struct pseudo_list *dominators;
469         int partial;

471         /* Unreachable load? Undo it */
472         if (!bb) {
473                 insn->opcode = OP_LNOP;
474                 return 1;
475         }

477         partial = 0;
478         FOR_EACH_PTR(bb->insns, one) {
479                 int dominance;
480                 if (one == insn)
481                         goto found;
482                 dominance = dominates(pseudo, insn, one, local);
483                 if (dominance < 0) {
484                         /* Ignore partial load dominators */
485                         if (one->opcode == OP_LOAD)
486                                 continue;
487                         dom = NULL;
488                         partial = 1;
489                         continue;
490                 }
491                 if (!dominance)
492                         continue;
493                 dom = one;
494                 partial = 0;
495         } END_FOR_EACH_PTR(one);
496         /* Whaa? */
497         warning(pseudo->sym->pos, "unable to find symbol read");
498         return 0;
499 found:
500         if (partial)
501                 return 0;

503         if (dom) {
504                 convert_load_instruction(insn, dom->target);
505                 return 1;
506         }

508         /* OK, go find the parents */
509         bb->generation = generation;

511         dominators = NULL;
512         if (!find_dominating_parents(pseudo, insn, bb, generation, &dominators,
513                 return 0;

515         /* This happens with initial assignments to structures etc.. */
516         if (!dominators) {
517                 if (!local)
518                         return 0;
519                 check_access(insn);
520                 convert_load_instruction(insn, value_pseudo(insn->type, 0));
521                 return 1;
522         }
```

```
524         /*
525          * If we find just one dominating instruction, we
526          * can turn it into a direct thing. Otherwise we'll
527          * have to turn the load into a phi-node of the
528          * dominators.
529          */
530         rewrite_load_instruction(insn, dominators);
531         return 1;
532 }

534 static void kill_store(struct instruction *insn)
535 {
536         if (insn) {
537                 insn->bb = NULL;
538                 insn->opcode = OP_SNOP;
539                 kill_use(&insn->target);
540         }
541 }

543 /* Kill a pseudo that is dead on exit from the bb */
544 static void kill_dead_stores(pseudo_t pseudo, unsigned long generation, struct b
545 {
546         struct instruction *insn;
547         struct basic_block *parent;

549         if (bb->generation == generation)
550                 return;
551         bb->generation = generation;
552         FOR_EACH_PTR_REVERSE(bb->insns, insn) {
553                 int opcode = insn->opcode;

555                 if (opcode != OP_LOAD && opcode != OP_STORE) {
556                         if (local)
557                                 continue;
558                         if (opcode == OP_CALL)
559                                 return;
560                         continue;
561                 }
562                 if (insn->src == pseudo) {
563                         if (opcode == OP_LOAD)
564                                 return;
565                         kill_store(insn);
566                         continue;
567                 }
568                 if (local)
569                         continue;
570                 if (insn->src->type != PSEUDO_SYM)
571                         return;
572         } END_FOR_EACH_PTR_REVERSE(insn);

574         FOR_EACH_PTR(bb->parents, parent) {
575                 struct basic_block *child;
576                 FOR_EACH_PTR(parent->children, child) {
577                         if (child && child != bb)
578                                 return;
579                 } END_FOR_EACH_PTR(child);
580                 kill_dead_stores(pseudo, generation, parent, local);
581         } END_FOR_EACH_PTR(parent);
582 }

584 /*
585  * This should see if the "insn" trivially dominates some previous store, and ki
586  * store if unnecessary.
587  */
588 static void kill_dominated_stores(pseudo_t pseudo, struct instruction *insn,
```



```
589         unsigned long generation, struct basic_block *bb, int local, int found)
590 {
591         struct instruction *one;
592         struct basic_block *parent;

594         /* Unreachable store? Undo it */
595         if (!bb) {
596                 kill_store(insn);
597                 return;
598         }
599         if (bb->generation == generation)
600                 return;
601         bb->generation = generation;
602         FOR_EACH_PTR_REVERSE(bb->insns, one) {
603                 int dominance;
604                 if (!found) {
605                         if (one != insn)
606                                 continue;
607                         found = 1;
608                         continue;
609                 }
610                 dominance = dominates(pseudo, insn, one, local);
611                 if (!dominance)
612                         continue;
613                 if (dominance < 0)
614                         return;
615                 if (one->opcode == OP_LOAD)
616                         return;
617                 kill_store(one);
618         } END_FOR_EACH_PTR_REVERSE(one);

620         if (!found) {
621                 warning(bb->pos, "Unable to find instruction");
622                 return;
623         }

625         FOR_EACH_PTR(bb->parents, parent) {
626                 struct basic_block *child;
627                 FOR_EACH_PTR(parent->children, child) {
628                         if (child && child != bb)
629                                 return;
630                 } END_FOR_EACH_PTR(child);
631                 kill_dominated_stores(pseudo, insn, generation, parent, local, f
632         } END_FOR_EACH_PTR(parent);
633 }

635 void check_access(struct instruction *insn)
636 {
637         pseudo_t pseudo = insn->src;

639         if (insn->bb && pseudo->type == PSEUDO_SYM) {
640                 int offset = insn->offset, bit = bytes_to_bits(offset) + insn->s
641                 struct symbol *sym = pseudo->sym;

643                 if (sym->bit_size > 0 && (offset < 0 || bit > sym->bit_size))
644                         warning(insn->pos, "invalid access %s '%s' (%d %d)",
645                                 offset < 0 ? "below" : "past the end of",
646                                 show_ident(sym->ident), offset,
647                                 bits_to_bytes(sym->bit_size));
648         }
649 }

651 static void simplify_one_symbol(struct entrypoint *ep, struct symbol *sym)
652 {
653         pseudo_t pseudo;
654         struct pseudo_user *pu;
```

```
655         unsigned long mod;
656         int all;

658         /* Never used as a symbol? */
659         pseudo = sym->pseudo;
660         if (!pseudo)
661                 return;

663         /* We don't do coverage analysis of volatiles.. */
664         if (sym->ctype.modifiers & MOD_VOLATILE)
665                 return;

667         /* ..and symbols with external visibility need more care */
668         mod = sym->ctype.modifiers & (MOD_NONLOCAL | MOD_STATIC | MOD_ADDRESSABL
669         if (mod)
670                 goto external_visibility;

672         FOR_EACH_PTR(pseudo->users, pu) {
673                 /* We know that the symbol-pseudo use is the "src" in the instru
674                 struct instruction *insn = pu->insn;

676                 switch (insn->opcode) {
677                 case OP_STORE:
678                         break;
679                 case OP_LOAD:
680                         break;
681                 case OP_SYMADDR:
682                         if (!insn->bb)
683                                 continue;
684                         mod |= MOD_ADDRESSABLE;
685                         goto external_visibility;
686                 case OP_NOP:
687                 case OP_SNOP:
688                 case OP_LNOP:
689                 case OP_PHI:
690                         continue;
691                 default:
692                         warning(sym->pos, "symbol '%s' pseudo used in unexpected
693                 }
694         } END_FOR_EACH_PTR(pu);

696 external_visibility:
697         all = 1;
698         FOR_EACH_PTR_REVERSE(pseudo->users, pu) {
699                 struct instruction *insn = pu->insn;
700                 if (insn->opcode == OP_LOAD)
701                         all &= find_dominating_stores(pseudo, insn, ++bb_generat
702         } END_FOR_EACH_PTR_REVERSE(pu);

704         /* If we converted all the loads, remove the stores. They are dead */
705         if (all && !mod) {
706                 FOR_EACH_PTR(pseudo->users, pu) {
707                         struct instruction *insn = pu->insn;
708                         if (insn->opcode == OP_STORE)
709                                 kill_store(insn);
710                 } END_FOR_EACH_PTR(pu);
711         } else {
712                 /*
713                  * If we couldn't take the shortcut, see if we can at least kill
714                  * of them..
715                  */
716                 FOR_EACH_PTR(pseudo->users, pu) {
717                         struct instruction *insn = pu->insn;
718                         if (insn->opcode == OP_STORE)
719                                 kill_dominated_stores(pseudo, insn, ++bb_generat
720                 } END_FOR_EACH_PTR(pu);
```



```
722                 if (!(mod & (MOD_NONLOCAL | MOD_STATIC))) {
723                         struct basic_block *bb;
724                         FOR_EACH_PTR(ep->bbs, bb) {
725                                 if (!bb->children)
726                                         kill_dead_stores(pseudo, ++bb_generation
727                         } END_FOR_EACH_PTR(bb);
728                 }
729         }
730
731         return;
732 }

734 void simplify_symbol_usage(struct entrypoint *ep)
735 {
736         pseudo_t pseudo;

738         FOR_EACH_PTR(ep->accesses, pseudo) {
739                 simplify_one_symbol(ep, pseudo->sym);
740         } END_FOR_EACH_PTR(pseudo);
741 }

743 static void mark_bb_reachable(struct basic_block *bb, unsigned long generation)
744 {
745         struct basic_block *child;

747         if (bb->generation == generation)
748                 return;
749         bb->generation = generation;
750         FOR_EACH_PTR(bb->children, child) {
751                 mark_bb_reachable(child, generation);
752         } END_FOR_EACH_PTR(child);
753 }

755 static void kill_defs(struct instruction *insn)
756 {
757         pseudo_t target = insn->target;

759         if (!has_use_list(target))
760                 return;
761         if (target->def != insn)
762                 return;

764         convert_instruction_target(insn, VOID);
765 }

767 void kill_bb(struct basic_block *bb)
768 {
769         struct instruction *insn;
770         struct basic_block *child, *parent;

772         FOR_EACH_PTR(bb->insns, insn) {
773                 kill_instruction_force(insn);
774                 kill_defs(insn);
775                 /*
776                  * We kill unreachable instructions even if they
777                  * otherwise aren't "killable" (e.g. volatile loads)
778                  */
779         } END_FOR_EACH_PTR(insn);
780         bb->insns = NULL;

782         FOR_EACH_PTR(bb->children, child) {
783                 remove_bb_from_list(&child->parents, bb, 0);
784         } END_FOR_EACH_PTR(child);
785         bb->children = NULL;
```

```
787         FOR_EACH_PTR(bb->parents, parent) {
788                 remove_bb_from_list(&parent->children, bb, 0);
789         } END_FOR_EACH_PTR(parent);
790         bb->parents = NULL;
791 }

793 void kill_unreachable_bbs(struct entrypoint *ep)
794 {
795         struct basic_block *bb;
796         unsigned long generation = ++bb_generation;

798         mark_bb_reachable(ep->entry->bb, generation);
799         FOR_EACH_PTR(ep->bbs, bb) {
800                 if (bb->generation == generation)
801                         continue;
802                 /* Mark it as being dead */
803                 kill_bb(bb);
804                 bb->ep = NULL;
805                 DELETE_CURRENT_PTR(bb);
806         } END_FOR_EACH_PTR(bb);
807         PACK_PTR_LIST(&ep->bbs);
808 }

810 static int rewrite_parent_branch(struct basic_block *bb, struct basic_block *old
811 {
812         int changed = 0;
813         struct instruction *insn = last_instruction(bb->insns);

815         if (!insn)
816                 return 0;

818         /* Infinite loops: let's not "optimize" them.. */
819         if (old == new)
820                 return 0;

822         switch (insn->opcode) {
823         case OP_CBR:
824                 changed |= rewrite_branch(bb, &insn->bb_false, old, new);
825                 /* fall through */
826         case OP_BR:
827                 changed |= rewrite_branch(bb, &insn->bb_true, old, new);
828                 assert(changed);
829                 return changed;
830         case OP_SWITCH: {
831                 struct multijmp *jmp;
832                 FOR_EACH_PTR(insn->multijmp_list, jmp) {
833                         changed |= rewrite_branch(bb, &jmp->target, old, new);
834                 } END_FOR_EACH_PTR(jmp);
835                 assert(changed);
836                 return changed;
837         }
838         default:
839                 return 0;
840         }
841 }

843 static struct basic_block * rewrite_branch_bb(struct basic_block *bb, struct ins
844 {
845         struct basic_block *parent;
846         struct basic_block *target = br->bb_true;
847         struct basic_block *false = br->bb_false;

849         if (br->opcode == OP_CBR) {
850                 pseudo_t cond = br->cond;
851                 if (cond->type != PSEUDO_VAL)
852                         return NULL;
```



```
853                 target = cond->value ? target : false;
854         }

856         /*
857          * We can't do FOR_EACH_PTR() here, because the parent list
858          * may change when we rewrite the parent.
859          */
860         while ((parent = first_basic_block(bb->parents)) != NULL) {
861                 if (!rewrite_parent_branch(parent, bb, target))
862                         return NULL;
863         }
864         return target;
865 }

867 static void vrfy_bb_in_list(struct basic_block *bb, struct basic_block_list *lis
868 {
869         if (bb) {
870                 struct basic_block *tmp;
871                 int no_bb_in_list = 0;

873                 FOR_EACH_PTR(list, tmp) {
874                         if (bb == tmp)
875                                 return;
876                 } END_FOR_EACH_PTR(tmp);
877                 assert(no_bb_in_list);
878         }
879 }

881 static void vrfy_parents(struct basic_block *bb)
882 {
883         struct basic_block *tmp;
884         FOR_EACH_PTR(bb->parents, tmp) {
885                 vrfy_bb_in_list(bb, tmp->children);
886         } END_FOR_EACH_PTR(tmp);
887 }

889 static void vrfy_children(struct basic_block *bb)
890 {
891         struct basic_block *tmp;
892         struct instruction *br = last_instruction(bb->insns);

894         if (!br) {
895                 assert(!bb->children);
896                 return;
897         }
898         switch (br->opcode) {
899                 struct multijmp *jmp;
900         case OP_CBR:
901                 vrfy_bb_in_list(br->bb_false, bb->children);
902                 /* fall through */
903         case OP_BR:
904                 vrfy_bb_in_list(br->bb_true, bb->children);
905                 break;
906         case OP_SWITCH:
907         case OP_COMPUTEDGOTO:
908                 FOR_EACH_PTR(br->multijmp_list, jmp) {
909                         vrfy_bb_in_list(jmp->target, bb->children);
910                 } END_FOR_EACH_PTR(jmp);
911                 break;
912         default:
913                 break;
914         }
915
916         FOR_EACH_PTR(bb->children, tmp) {
917                 vrfy_bb_in_list(bb, tmp->parents);
918         } END_FOR_EACH_PTR(tmp);
```

```
}

static void vrfy_bb_flow(struct basic_block *bb)
{
        vrfy_children(bb);
        vrfy_parents(bb);
}

void vrfy_flow(struct entrypoint *ep)
{
        struct basic_block *bb;
        struct basic_block *entry = ep->entry->bb;

        FOR_EACH_PTR(ep->bbs, bb) {
                if (bb == entry)
                        entry = NULL;
                vrfy_bb_flow(bb);
        } END_FOR_EACH_PTR(bb);
        assert(!entry);
}

void pack_basic_blocks(struct entrypoint *ep)
{
        struct basic_block *bb;

        /* See if we can merge a bb into another one.. */
        FOR_EACH_PTR(ep->bbs, bb) {
                struct instruction *first, *insn;
                struct basic_block *parent, *child, *last;

                if (!bb_reachable(bb))
                        continue;

                /*
                 * Just a branch?
                 */
                FOR_EACH_PTR(bb->insns, first) {
                        if (!first->bb)
                                continue;
                        switch (first->opcode) {
                        case OP_NOP: case OP_LNOP: case OP_SNOP:
                                continue;
                        case OP_CBR:
                        case OP_BR: {
                                struct basic_block *replace;
                                replace = rewrite_branch_bb(bb, first);
                                if (replace) {
                                        kill_bb(bb);
                                        goto no_merge;
                                }
                        }
                        /* fallthrough */
                        default:
                                goto out;
                        }
                } END_FOR_EACH_PTR(first);

out:
                /*
                 * See if we only have one parent..
                 */
                last = NULL;
                FOR_EACH_PTR(bb->parents, parent) {
                        if (last) {
                                if (last != parent)
                                        goto no_merge;
```



```
                                        continue;
                                }
                                last = parent;
                        } END_FOR_EACH_PTR(parent);

                        parent = last;
                        if (!parent || parent == bb)
                                continue;

                        /*
                         * Goodie. See if the parent can merge..
                         */
                        FOR_EACH_PTR(parent->children, child) {
                                if (child != bb)
                                        goto no_merge;
                        } END_FOR_EACH_PTR(child);

                        /*
                         * Merge the two.
                         */
                        repeat_phase |= REPEAT_CSE;

                        parent->children = bb->children;
                        bb->children = NULL;
                        bb->parents = NULL;

                        FOR_EACH_PTR(parent->children, child) {
                                replace_bb_in_list(&child->parents, bb, parent, 0);
                        } END_FOR_EACH_PTR(child);

                        kill_instruction(delete_last_instruction(&parent->insns));
                        FOR_EACH_PTR(bb->insns, insn) {
                                if (insn->bb) {
                                        assert(insn->bb == bb);
                                        insn->bb = parent;
                                }
                                add_instruction(&parent->insns, insn);
                        } END_FOR_EACH_PTR(insn);
                        bb->insns = NULL;

                no_merge:
                        /* nothing to do */;
                } END_FOR_EACH_PTR(bb);
}
```

```
**********************************************************
    1606 Fri Dec 21 15:00:12 2018
new/usr/src/tools/smatch/src/flow.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef FLOW_H
#define FLOW_H

#include "lib.h"

extern unsigned long bb_generation;

#define REPEAT_CSE              1
#define REPEAT_SYMBOL_CLEANUP   2
#define REPEAT_CFG_CLEANUP      3

struct entrypoint;
struct instruction;

extern int simplify_flow(struct entrypoint *ep);

extern void simplify_symbol_usage(struct entrypoint *ep);
extern void simplify_memops(struct entrypoint *ep);
extern void pack_basic_blocks(struct entrypoint *ep);

extern void convert_instruction_target(struct instruction *insn, pseudo_t src);
extern void cleanup_and_cse(struct entrypoint *ep);
extern int simplify_instruction(struct instruction *);

extern void kill_bb(struct basic_block *);
extern void kill_use(pseudo_t *);
extern void kill_unreachable_bbs(struct entrypoint *ep);

extern void kill_insn(struct instruction *, int force);
static inline void kill_instruction(struct instruction *insn)
{
        kill_insn(insn, 0);
}
static inline void kill_instruction_force(struct instruction *insn)
{
        kill_insn(insn, 1);
}

void check_access(struct instruction *insn);
void convert_load_instruction(struct instruction *, pseudo_t);
void rewrite_load_instruction(struct instruction *, struct pseudo_list *);
int dominates(pseudo_t pseudo, struct instruction *insn, struct instruction *dom

extern void clear_liveness(struct entrypoint *ep);
extern void track_pseudo_liveness(struct entrypoint *ep);
extern void track_pseudo_death(struct entrypoint *ep);
extern void track_phi_uses(struct instruction *insn);

extern void vrfy_flow(struct entrypoint *ep);
extern int pseudo_in_list(struct pseudo_list *list, pseudo_t pseudo);

#endif
```

```
**********************************************************
    4702 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/gcc-attr-list.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
GCC_ATTR(BELOW100)
GCC_ATTR(OS_Task)
GCC_ATTR(OS_main)
GCC_ATTR(OS_task)
GCC_ATTR(abi_tag)
GCC_ATTR(absdata)
GCC_ATTR(address)
GCC_ATTR(alias)
GCC_ATTR(aligned)
GCC_ATTR(alloc_align)
GCC_ATTR(alloc_size)
GCC_ATTR(altivec)
GCC_ATTR(always_inline)
GCC_ATTR(artificial)
GCC_ATTR(assume_aligned)
GCC_ATTR(aux)
GCC_ATTR(bank_switch)
GCC_ATTR(based)
GCC_ATTR(below100)
GCC_ATTR(bnd_instrument)
GCC_ATTR(bnd_legacy)
GCC_ATTR(bnd_variable_size)
GCC_ATTR(break_handler)
GCC_ATTR(brk_interrupt)
GCC_ATTR(callee_pop_aggregate_return)
GCC_ATTR(cb)
GCC_ATTR(cdecl)
GCC_ATTR(cleanup)
GCC_ATTR(cmse_nonsecure_call)
GCC_ATTR(cmse_nonsecure_entry)
GCC_ATTR(cold)
GCC_ATTR(common)
GCC_ATTR(common_object)
GCC_ATTR(const)
GCC_ATTR(constructor)
GCC_ATTR(critical)
GCC_ATTR(default)
GCC_ATTR(deprecated)
GCC_ATTR(designated_init)
GCC_ATTR(destructor)
GCC_ATTR(disinterrupt)
GCC_ATTR(dllexport)
GCC_ATTR(dllimport)
GCC_ATTR(eightbit_data)
GCC_ATTR(either)
GCC_ATTR(error)
GCC_ATTR(exception)
GCC_ATTR(exception_handler)
GCC_ATTR(externally_visible)
GCC_ATTR(fallthrough)
GCC_ATTR(far)
GCC_ATTR(fast_interrupt)
GCC_ATTR(fastcall)
GCC_ATTR(flatten)
GCC_ATTR(force_align_arg_pointer)
GCC_ATTR(format)
GCC_ATTR(format_arg)
GCC_ATTR(forwarder_section)
GCC_ATTR(function_return)
GCC_ATTR(function_return_mem)
```



```
GCC_ATTR(function_return_reg)
GCC_ATTR(function_vector)
GCC_ATTR(gcc_struct)
GCC_ATTR(gnu_inline)
GCC_ATTR(hidden)
GCC_ATTR(hot)
GCC_ATTR(hotpatch)
GCC_ATTR(ifunc)
GCC_ATTR(indirect_branch)
GCC_ATTR(indirect_branch_call)
GCC_ATTR(indirect_branch_jump)
GCC_ATTR(init_priority)
GCC_ATTR(interfacearm)
GCC_ATTR(internal)
GCC_ATTR(interrupt)
GCC_ATTR(interrupt_handler)
GCC_ATTR(interrupt_thread)
GCC_ATTR(io)
GCC_ATTR(io_low)
GCC_ATTR(isr)
GCC_ATTR(jli_always)
GCC_ATTR(jli_fixed)
GCC_ATTR(keep_interrupts_masked)
GCC_ATTR(kernel)
GCC_ATTR(kspisusp)
GCC_ATTR(l1_data)
GCC_ATTR(l1_data_A)
GCC_ATTR(l1_data_B)
GCC_ATTR(l1_text)
GCC_ATTR(l2)
GCC_ATTR(leaf)
GCC_ATTR(long_call)
GCC_ATTR(longcall)
GCC_ATTR(lower)
GCC_ATTR(malloc)
GCC_ATTR(may_alias)
GCC_ATTR(maybe_unused)
GCC_ATTR(medium_call)
GCC_ATTR(micromips)
GCC_ATTR(mips16)
GCC_ATTR(mode)
GCC_ATTR(model)
GCC_ATTR(monitor)
GCC_ATTR(ms_abi)
GCC_ATTR(ms_hook_prologue)
GCC_ATTR(ms_struct)
GCC_ATTR(naked)
GCC_ATTR(near)
GCC_ATTR(nested)
GCC_ATTR(nested_ready)
GCC_ATTR(nesting)
GCC_ATTR(nmi)
GCC_ATTR(nmi_handler)
GCC_ATTR(no_address_safety_analysis)
GCC_ATTR(no_caller_saved_registers)
GCC_ATTR(no_gccisr)
GCC_ATTR(no_icf)
GCC_ATTR(no_instrument_function)
GCC_ATTR(no_profile_instrument_function)
GCC_ATTR(no_reorder)
GCC_ATTR(no_sanitize)
GCC_ATTR(no_sanitize_address)
GCC_ATTR(no_sanitize_thread)
GCC_ATTR(no_sanitize_undefined)
GCC_ATTR(no_split_stack)
GCC_ATTR(no_stack_limit)
```

```
127 GCC_ATTR(nocf_check)
128 GCC_ATTR(noclone)
129 GCC_ATTR(nocommon)
130 GCC_ATTR(nocompression)
131 GCC_ATTR(nodiscard)
132 GCC_ATTR(noinit)
133 GCC_ATTR(noinline)
134 GCC_ATTR(noipa)
135 GCC_ATTR(nomicromips)
136 GCC_ATTR(nomips16)
137 GCC_ATTR(nonnull)
138 GCC_ATTR(nonstring)
139 GCC_ATTR(noplt)
140 GCC_ATTR(noreturn)
141 GCC_ATTR(nosave_low_regs)
142 GCC_ATTR(not_nested)
143 GCC_ATTR(nothrow)
144 GCC_ATTR(notshared)
145 GCC_ATTR(optimize)
146 GCC_ATTR(packed)
147 GCC_ATTR(partial_save)
148 GCC_ATTR(patchable_function_entry)
149 GCC_ATTR(pcs)
150 GCC_ATTR(persistent)
151 GCC_ATTR(progmem)
152 GCC_ATTR(protected)
153 GCC_ATTR(pure)
154 GCC_ATTR(reentrant)
155 GCC_ATTR(regparm)
156 GCC_ATTR(renesas)
157 GCC_ATTR(resbank)
158 GCC_ATTR(reset)
159 GCC_ATTR(returns_nonnull)
160 GCC_ATTR(returns_twice)
161 GCC_ATTR(s390_vector_bool)
162 GCC_ATTR(saddr)
163 GCC_ATTR(save_all)
164 GCC_ATTR(save_volatiles)
165 GCC_ATTR(saveall)
166 GCC_ATTR(scalar_storage_order)
167 GCC_ATTR(sda)
168 GCC_ATTR(section)
169 GCC_ATTR(secure_call)
170 GCC_ATTR(selectany)
171 GCC_ATTR(sentinel)
172 GCC_ATTR(shared)
173 GCC_ATTR(short_call)
174 GCC_ATTR(shortcall)
175 GCC_ATTR(signal)
176 GCC_ATTR(simd)
177 GCC_ATTR(sp_switch)
178 GCC_ATTR(spu_vector)
179 GCC_ATTR(sseregparm)
180 GCC_ATTR(stack_protect)
181 GCC_ATTR(stdcall)
182 GCC_ATTR(syscall_linkage)
183 GCC_ATTR(sysv_abi)
184 GCC_ATTR(target)
185 GCC_ATTR(target_clones)
186 GCC_ATTR(tda)
187 GCC_ATTR(thiscall)
188 GCC_ATTR(tiny)
189 GCC_ATTR(tiny_data)
190 GCC_ATTR(tls_model)
191 GCC_ATTR(transaction_callable)
192 GCC_ATTR(transaction_may_cancel_outer)
```



```
193 GCC_ATTR(transaction_pure)
194 GCC_ATTR(transaction_safe)
195 GCC_ATTR(transaction_safe_dynamic)
196 GCC_ATTR(transaction_unsafe)
197 GCC_ATTR(transaction_wrap)
198 GCC_ATTR(transparent_union)
199 GCC_ATTR(trap_exit)
200 GCC_ATTR(trapa_handler)
201 GCC_ATTR(uncached)
202 GCC_ATTR(unused)
203 GCC_ATTR(upper)
204 GCC_ATTR(use_debug_exception_return)
205 GCC_ATTR(use_shadow_register_set)
206 GCC_ATTR(used)
207 GCC_ATTR(vector)
208 GCC_ATTR(vector_size)
209 GCC_ATTR(version_id)
210 GCC_ATTR(visibility)
211 GCC_ATTR(vliw)
212 GCC_ATTR(volatile)
213 GCC_ATTR(wakeup)
214 GCC_ATTR(warm)
215 GCC_ATTR(warn_if_not_aligned)
216 GCC_ATTR(warn_unused)
217 GCC_ATTR(warn_unused_result)
218 GCC_ATTR(warning)
219 GCC_ATTR(weak)
220 GCC_ATTR(weakref)
221 GCC_ATTR(zda)
```

```
**********************************************************
    5578 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/gdbhelpers
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
# Don't forget to rebuild sparse with uncommented debug options
# in the Makefile. Also, gcc 3 is known to screw up with the
# cpp macros in the debugging info.


# If a gdb_show_* function is running at a non-zero recursion
# level, only a short summary is shown, preventing further
# recursion. Also note that gdb has only one, global, scope
# for variables, so we need to be careful with recursions.


set $showing_token = 0
set $showing_ident = 0
set $showing_symbol = 0

set $ntabs = 0

define gdb_tabs
        set $tmp = $ntabs
        while ($tmp--)
                printf "\t"
        end
end


# Ptr list handling
define ptr_entry
        set $ptr = $arg0
        set $index = $arg1
        set $entry = &($arg2)

        set *($entry) = (void *) (~3UL & (unsigned long)$ptr->list[$index])
end


# Ptr list looping skeleton
define gdb_ptr_list_for_each

        set $my_head = (struct ptr_list *) $arg0
        set $my_list = $my_head


        if ($my_head)
                while (1)
                        set $my_nr = 0
                        while ($my_nr < $my_list->nr)

                                # Put your iterator code here
                                set $my_nr++
                        end

                        if (($my_list = $my_list->next) == $my_head)
                                loop_break
                        end
                end
        end
end

# Show symbols in a symbol_list. Non-recursive
```



```
define gdb_ptr_list_for_each_show_symbol

        set $my_head = (struct ptr_list *) $arg0
        set $my_list = $my_head


        if ($my_head)
                while (1)
                        set $my_nr = 0
                        while ($my_nr < ($my_list)->nr)
                                set $my_symbol = (struct symbol *) ((~3UL) & (un
                                gdb_show_symbol($my_symbol)

                                set $my_nr++
                        end

                        set $my_list = ($my_list)->next
                        if ($my_list == $my_head)
                                loop_break
                        end
                end
        end
end


#define gdb_show_statement


# Recursive
define gdb_show_ctype
        printf "modifiers: "
        if ($arg0->modifiers & MOD_AUTO)
                printf "MOD_AUTO "
        end
        if ($arg0->modifiers & MOD_REGISTER)
                printf "MOD_REGISTER "
        end
        if ($arg0->modifiers & MOD_STATIC)
                printf "MOD_STATIC "
        end
        if ($arg0->modifiers & MOD_EXTERN)
                printf "MOD_EXTERN "
        end
        if ($arg0->modifiers & MOD_CONST)
                printf "MOD_CONST "
        end
        if ($arg0->modifiers & MOD_VOLATILE)
                printf "MOD_VOLATILE "
        end
        if ($arg0->modifiers & MOD_SIGNED)
                printf "MOD_SIGNED "
        end
        if ($arg0->modifiers & MOD_UNSIGNED)
                printf "MOD_UNSIGNED "
        end
        if ($arg0->modifiers & MOD_CHAR)
                printf "MOD_CHAR "
        end
        if ($arg0->modifiers & MOD_SHORT)
                printf "MOD_SHORT "
        end
        if ($arg0->modifiers & MOD_LONG)
                printf "MOD_LONG "
        end
        if ($arg0->modifiers & MOD_LONGLONG)
                printf "MOD_LONGLONG "
```

```
127         end
128         if ($arg0->modifiers & MOD_LONGLONGLONG)
129                 printf "MOD_LONGLONGLONG "
130         end
131         if ($arg0->modifiers & MOD_TYPEDEF)
132                 printf "MOD_TYPEDEF "
133         end
134         if ($arg0->modifiers & MOD_INLINE)
135                 printf "MOD_INLINE "
136         end
137         if ($arg0->modifiers & MOD_ADDRESSABLE)
138                 printf "MOD_ADDRESSABLE "
139         end
140         if ($arg0->modifiers & MOD_NOCAST)
141                 printf "MOD_NOCAST "
142         end
143         if ($arg0->modifiers & MOD_NODEREF)
144                 printf "MOD_NODEREF "
145         end
146         if ($arg0->modifiers & MOD_ACCESSED)
147                 printf "MOD_ACCESSED "
148         end
149         if ($arg0->modifiers & MOD_TOPLEVEL)
150                 printf "MOD_TOPLEVEL "
151         end
152         if ($arg0->modifiers & MOD_ASSIGNED)
153                 printf "MOD_ASSIGNED "
154         end
155         if ($arg0->modifiers & MOD_TYPE)
156                 printf "MOD_TYPE "
157         end
158         if ($arg0->modifiers & MOD_SAFE)
159                 printf "MOD_SAFE "
160         end
161         if ($arg0->modifiers & MOD_USERTYPE)
162                 printf "MOD_USERTYPE "
163         end
164         if ($arg0->modifiers & MOD_EXPLICITLY_SIGNED)
165                 printf "MOD_EXPLICITLY_SIGNED"
166         end
167         if ($arg0->modifiers & MOD_BITWISE)
168                 printf "MOD_BITWISE "
169         end
170         if (!$arg0->modifiers)
171                 printf "0"
172         end

174         printf ", alignment = %d", $arg0->alignment
175         if ($arg0->as)
176                 printf ", address_space = %d", $arg0->as
177         end
178         printf "\n"


181         set $ntabs++
182         if ($arg0->base_type)
183                 gdb_tabs
184                 printf "base_type = "
185                 gdb_show_symbol($arg0->base_type)
186         end

188         set $ntabs--


191 end
```



```
193 define gdb_show_symbol
194         printf "(%x) type = ", $arg0
195         output $arg0->type
196         printf ", namespace = "
197         output $arg0->namespace
198         if ($arg0->ident)
199                 printf ", ident = %s\n", show_ident($arg0->ident)
200         else
201                 printf ", ident = NULL\n"
202         end

204 #       print "zz"

206         gdb_tabs
207         printf "dump:\n"
208         call show_symbol($arg0)

210         set $ntabs++
211         if ($arg0->namespace == NS_SYMBOL)
212                 gdb_tabs
213                 printf "ctype = "
214                 gdb_show_ctype(&($arg0->ctype))
215         end
216         set $ntabs--
217 end


220 # non-recursive
221 define gdb_show_symbols_next_id
222         set $sym = $arg0
223         printf "{\n"
224         set $ntabs++
225         while ($sym)
226                 gdb_tabs
227                 printf "symbol = "
228                 gdb_show_symbol($sym)
229                 set $sym = $sym->next_id
230         end
231         set $ntabs--
232         gdb_tabs
233         printf "}\n"
234 end

236 define gdb_show_ident
237         if ($arg0)
238                 printf "(%p) '%s'\n", $arg0, show_ident($arg0)
239         else
240                 printf "NULL\n"
241         end

243         if (! $showing_ident)
244                 set $showing_ident = 1
245                 set $ntabs++

247                 set $ident = $arg0

249                 if ($ident->symbols)
250                         gdb_tabs
251                         printf "symbols = "
252                         gdb_show_symbols_next_id($ident->symbols)
253                 end

255                 set $ntabs--
256                 set $showing_ident = 0
257         end
258 end
```

```
260 define gdb_show_token
261         printf "%p: '%s', type = ", $arg0, show_token($arg0)
262         output (enum token_type) ($arg0)->pos.type
263         printf "\n"

265         if (! $showing_token)
266                 set $showing_token = 1
267                 set $ntabs++

269                 set $token = $arg0

271                 if ($token->pos.type == TOKEN_IDENT)
272                         gdb_tabs
273                         printf "ident = "
274                         gdb_show_ident $token.ident
275                 end

277                 if ($token->pos.type == TOKEN_MACRO_ARGUMENT)
278                         gdb_tabs
279                         printf "argnum = %d\n", $token->argnum
280                 end

282                 if ($token->pos.type == TOKEN_SPECIAL)
283                         gdb_tabs
284                         printf "special = \"%s\"\n", show_special($token.special
285                 end

287                 set $ntabs--
288                 set $showing_token = 0
289         end
290 end

292 # non-recursive
293 define gdb_show_tokens
294         set $t = $arg0
295         printf "{\n"
296         set $ntabs++
297         while ($t != &eof_token_entry)
298                 gdb_tabs
299                 printf "token = "
300                 gdb_show_token($t)
301                 set $t = ($t)->next
302         end
303         set $ntabs--
304         gdb_tabs
305         printf "}\n"
306 end
```

```
**********************************************************
    5766 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/graph.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Copyright ´' International Business Machines Corp., 2006
 *              Adelard LLP, 2007
 *
 * Author: Josh Triplett <josh@freedesktop.org>
 *         Dan Sheridan <djs@adelard.com>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"
#include "linearize.h"


/* Draw the subgraph for a given entrypoint. Includes details of loads
 * and stores for globals, and marks return bbs */
static void graph_ep(struct entrypoint *ep)
{
        struct basic_block *bb;
        struct instruction *insn;

        const char *fname, *sname;

        fname = show_ident(ep->name->ident);
        sname = stream_name(ep->entry->bb->pos.stream);

        printf("subgraph cluster%p {\n"
               "    color=blue;\n"
               "    label=<<TABLE BORDER=\"0\" CELLBORDER=\"0\">\n"
               "             <TR><TD>%s</TD></TR>\n"
               "             <TR><TD><FONT POINT-SIZE=\"21\">%s()</FONT></TD></T
               "           </TABLE>>;\n"
               "    file=\"%s\";\n"
```



```
               "    fun=\"%s\";\n"
               "    ep=bb%p;\n",
               ep, sname, fname, sname, fname, ep->entry->bb);

        FOR_EACH_PTR(ep->bbs, bb) {
                struct basic_block *child;
                int ret = 0;
                const char * s = ", ls=\"[";

                /* Node for the bb */
                printf("    bb%p [shape=ellipse,label=%d,line=%d,col=%d",
                       bb, bb->pos.line, bb->pos.line, bb->pos.pos);


                /* List loads and stores */
                FOR_EACH_PTR(bb->insns, insn) {
                        switch(insn->opcode) {
                        case OP_STORE:
                                if (insn->symbol->type == PSEUDO_SYM) {
                                  printf("%s store(%s)", s, show_ident(insn->sym
                                  s = ",";
                                }
                                break;

                        case OP_LOAD:
                                if (insn->symbol->type == PSEUDO_SYM) {
                                  printf("%s load(%s)", s, show_ident(insn->symb
                                  s = ",";
                                }
                                break;

                        case OP_RET:
                                ret = 1;
                                break;

                        }
                } END_FOR_EACH_PTR(insn);
                if (s[1] == 0)
                        printf("]\"");
                if (ret)
                        printf(",op=ret");
                printf("];\n");

                /* Edges between bbs; lower weight for upward edges */
                FOR_EACH_PTR(bb->children, child) {
                        printf("    bb%p -> bb%p [op=br, %s];\n", bb, child,
                               (bb->pos.line > child->pos.line) ? "weight=5" : "
                } END_FOR_EACH_PTR(child);
        } END_FOR_EACH_PTR(bb);

        printf("}\n");
}


/* Insert edges for intra- or inter-file calls, depending on the value
 * of internal. Bold edges are used for calls with destinations;
 * dashed for calls to external functions */
static void graph_calls(struct entrypoint *ep, int internal)
{
        struct basic_block *bb;
        struct instruction *insn;

        show_ident(ep->name->ident);
        stream_name(ep->entry->bb->pos.stream);

        FOR_EACH_PTR(ep->bbs, bb) {
```

```
127                 if (!bb)
128                         continue;
129                 if (!bb->parents && !bb->children && !bb->insns && verbose < 2)
130                         continue;

132                 FOR_EACH_PTR(bb->insns, insn) {
133                         if (insn->opcode == OP_CALL &&
134                             internal == !(insn->func->sym->ctype.modifiers & MOD

136                                 /* Find the symbol for the callee's definition *
137                                 struct symbol * sym;
138                                 if (insn->func->type == PSEUDO_SYM) {
139                                         for (sym = insn->func->sym->ident->symbo
140                                              sym; sym = sym->next_id) {
141                                                 if (sym->namespace & NS_SYMBOL &
142                                                         break;
143                                         }

145                                         if (sym)
146                                                 printf("bb%p -> bb%p"
147                                                        "[label=%d,line=%d,col=%d
148                                                        bb, sym->ep->entry->bb,
149                                                        insn->pos.line, insn->pos
150                                         else
151                                                 printf("bb%p -> \"%s\" "
152                                                        "[label=%d,line=%d,col=%d
153                                                        bb, show_pseudo(insn->fun
154                                                        insn->pos.line, insn->pos
155                                 }
156                         }
157                 } END_FOR_EACH_PTR(insn);
158         } END_FOR_EACH_PTR(bb);
159 }

161 int main(int argc, char **argv)
162 {
163         struct string_list *filelist = NULL;
164         char *file;
165         struct symbol *sym;

167         struct symbol_list *fsyms, *all_syms=NULL;

169         printf("digraph call_graph {\n");
170         fsyms = sparse_initialize(argc, argv, &filelist);
171         concat_symbol_list(fsyms, &all_syms);

173         /* Linearize all symbols, graph internal basic block
174          * structures and intra-file calls */
175         FOR_EACH_PTR_NOTAG(filelist, file) {

177                 fsyms = sparse(file);
178                 concat_symbol_list(fsyms, &all_syms);

180                 FOR_EACH_PTR(fsyms, sym) {
181                         expand_symbol(sym);
182                         linearize_symbol(sym);
183                 } END_FOR_EACH_PTR(sym);

185                 FOR_EACH_PTR(fsyms, sym) {
186                         if (sym->ep) {
187                                 graph_ep(sym->ep);
188                                 graph_calls(sym->ep, 1);
189                         }
190                 } END_FOR_EACH_PTR_NOTAG(sym);

192         } END_FOR_EACH_PTR_NOTAG(file);
```



```
194         /* Graph inter-file calls */
195         FOR_EACH_PTR(all_syms, sym) {
196                 if (sym->ep)
197                         graph_calls(sym->ep, 0);
198         } END_FOR_EACH_PTR_NOTAG(sym);

200         printf("}\n");
201         return 0;
202 }
```

```
**********************************************************
    2143 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/gvpr/return-paths
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/gvpr -f
// Split call sites into call site and return site nodes and add
// return edges
//
// Run with graph ... | return-paths

BEGIN {
        // Find the immediate parent subgraph of this object
        graph_t find_owner(obj_t o, graph_t g)
        {
                graph_t g1;
                for (g1 = fstsubg(g); g1; g1 = nxtsubg(g1))
                        if(isIn(g1,o)) return g1;
                return NULL;
        }
}

BEG_G {
        node_t calls[]; // Crude hash table for tracking who calls what
        graph_t g,g2;
        edge_t e,e2;
        string idx;
        node_t n, n2;
        int i;

        $tvtype = TV_en;
}

// Each call edge which hasn't already been seen
E [op == "call" && tail.split != 1] {
        int offset=0;

        // Clear the label of this call
        label = "";
        g = find_owner(tail, $G);

        // Consider each outgoing call. Split the node accordingly
        n = tail;
        for (e = fstout(tail); e; e = nxtout(e)) {
                if (e.op == "call") {

                        // Split node
                        n2 = node(g, sprintf("%s%s%d", tail.name, "x", offset++)
                        copyA(tail, n2);
                        n2.line = e.line;
                        n2.label = e.line;
                        n2.col = e.col;
                        n2.split = 1;
                        n2.op = "target";

                        // Record this call
                        g2 = find_owner(e.head, $G);
                        i = 0;
                        while(calls[sprintf("%s%d", g2.name, ++i)]) {
                        }
                        calls[sprintf("%s%d", g2.name, i)] = n2;

                        // Connect original to split
                        e2 = edge(n, n2, "call");
                        e2.style = "dotted";
```



```
                        e2.weight = 50;

                        // Replace this outedge
                        if (n != tail) {
                                e2 = edge(n, e.head, "transformed-call");
                                copyA(e,e2);
                                e2.label = "";
                                delete($G,e);
                        }

                        // Record where we were
                        n = n2;
                }
        }

        // Consider the outgoing control flow: move down to the bottom of
        // the call sequence nodes
        for (e = fstout(tail); e; e = nxtout(e)) {
                if (e.op == "br") {
                        // Replace this outedge
                        e2 = edge(n,e.head,"transformed");
                        copyA(e,e2);
                        delete($G,e);
                }
        }
}

// Each return node: add edges back to the caller
N [op == "ret"] {
        for (g = fstsubg($G); g; g = nxtsubg(g)) {
                if(isIn(g,$)) {
                        i = 0;
                        while(calls[sprintf("%s%d", g.name, ++i)]) {
                                e = edge($, calls[sprintf("%s%d", g.name, i)], "
                                e.style = "dotted";
                                e.op = "ret";
                                e.line = e.tail.line;
                                e.weight = 5;
                        }
                }
        }
}


END_G {
        write($);
}
```

```
**********************************************************
    1639 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/gvpr/subg-fwd
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/gvpr -f
// Compute the forward partition of the chosen function
//
// Run with graph ... | return-paths | subg-fwd -a functionname
//       or graph ... | subg-fwd


BEGIN {
        // Find the immediate parent subgraph of this object
        graph_t find_owner(obj_t o, graph_t g)
        {
                graph_t g1;
                for (g1 = fstsubg(g); g1; g1 = nxtsubg(g1))
                        if(isIn(g1,o)) return g1;
                return NULL;
        }
}

BEG_G {
        graph_t sg = subg ($, sprintf("incoming-%s", ARGV[0]));
        graph_t returns = graph("return-edges", ""); // Temporary graph to hold
        graph_t target, g, g2;
        node_t n;
        edge_t e;
        int i;

        $tvtype = TV_fwd;

        // find the ep corresponding to ARG[0]
        for (g = fstsubg($G); g; g = nxtsubg(g)) {
                if(g.fun == ARGV[0]) {
                        n = node($,g.ep);
                        $tvroot = n;
                        n.style = "filled";
                        target = g;
                        break;
                }
        }
        if(!target) {
                printf(2, "Function %s not found\n", ARGV[0]);
                exit(1);
        }
}

// Preserve external functions
E [op == "extern"] {
        subnode (sg, head);
}

// Move unused return edges into a separate graph so they don't get followed
N [op == "ret"] {
        for (e = fstout($); e; e = nxtout(e))
                if (e.op == "ret" && !isIn(sg, e.head)) {
                        clone(returns, e);
                        delete($G, e);
                }
}

// Recover elided return edge for this target node
N [op == "target" && indegree == 1] {
```



```
        n = copy(returns, $);
        e = fstin(n); // each target node can only have one return edge
        e = edge(copy(sg, e.tail), $, "recovered"); // clone should work here, b
        copyA(fstin(n), e);
}

// Copy relevant nodes
N {
        $tvroot = NULL;

        g = find_owner($, $G);
        if(g && g != sg)
                subnode (copy(sg, g), $);
}

END_G {
        induce(sg);
        write(sg);
}
```

```
**********************************************************
    1943 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/gvpr/subg-rev
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/gvpr -f
// Compute the reverse partition of the chosen function
//
// Run with graph ... | return-paths | subg-rev -a functionname


BEGIN {
        // Find the immediate parent subgraph of this object
        graph_t find_owner(obj_t o, graph_t g)
        {
                graph_t g1;
                for (g1 = fstsubg(g); g1; g1 = nxtsubg(g1))
                        if(isIn(g1,o)) return g1;
                return NULL;
        }
}

BEG_G {
        graph_t calls[]; // Crude hash table for tracking who calls what
        graph_t sg = subg ($, "reachable");
        graph_t target, g, g2;
        edge_t e;
        int i;

        $tvtype = TV_rev;

        // find the ep corresponding to ARG[0]
        for (g = fstsubg($G); g; g = nxtsubg(g)) {
                if(g.fun == ARGV[0]) {
                        node_t n;
                        n = node($,g.ep);
                        $tvroot = n;
                        n.style = "filled";
                        target = g;
                        break;
                }
        }
        if(!target) {
                printf(2, "Function %s not found\n", ARGV[0]);
                exit(1);
        }

        // Add the incoming call edges to the allowed call list
        i = 0;
        for(e = fstin(n); e; e = nxtin(e)) {
                if (e.op = "call") {
                        g2 = find_owner(e.tail, $G);
                        calls[sprintf("%s%d", g2.name, ++i)] = g;
                }
        }
}


E [op == "ret"] {

        // This is a return edge. Allow the corresponding call
        g = find_owner(head, $G);
        i = 0;
        while(calls[sprintf("%s%d", g.name, ++i)]) {
        }
```



```
        calls[sprintf("%s%d", g.name, i)] = find_owner(tail, $G);
        g2 = find_owner(tail, $G);
}


N [split == 1] {

        // Ignore returns back to the target function
        for (e = fstin($); e; e = nxtin(e))
                if (e.op == "ret" && isIn(target,e.tail))
                        delete($G,e);
}

N {
        $tvroot = NULL;

        for (e = fstin($); e; e = nxtin(e)) {
                if (e.op == "call") {
                        int found = 0;
                        g = find_owner(e.tail, $G);
                        i = 0;
                        while(calls[sprintf("%s%d", g.name, ++i)]) {
                                if (isIn(calls[sprintf("%s%d", g.name, i)],e.hea
                                        found = 1;
                        }
                        g2 = find_owner(e.head, $G);
                        if (!found) delete($G, e);
                }
        }

        for (g = fstsubg($G); g; g = nxtsubg(g)) {
                if(isIn(g,$) && g != sg) {
                        subnode (copy(sg, g), $);
                }
        }
}

END_G {
        induce(sg);
        write(sg);
}
```

```
**********************************************************
    2333 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/ident-list.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define IDENT(n) __IDENT(n## _ident, #n, 0)
#define IDENT_RESERVED(n) __IDENT(n## _ident, #n, 1)

/* Basic C reserved words.. */
IDENT_RESERVED(sizeof);
IDENT_RESERVED(if);
IDENT_RESERVED(else);
IDENT_RESERVED(return);
IDENT_RESERVED(switch);
IDENT_RESERVED(case);
IDENT_RESERVED(default);
IDENT_RESERVED(break);
IDENT_RESERVED(continue);
IDENT_RESERVED(for);
IDENT_RESERVED(while);
IDENT_RESERVED(do);
IDENT_RESERVED(goto);

/* C typenames. They get marked as reserved when initialized */
IDENT(struct);
IDENT(union);
IDENT(enum);
IDENT(__attribute); IDENT(__attribute__);
IDENT(volatile); IDENT(__volatile); IDENT(__volatile__);
IDENT(double);

/* C storage classes. They get marked as reserved when initialized */
IDENT(static);

/* C99 keywords */
IDENT(restrict); IDENT(__restrict); IDENT(__restrict__);
IDENT(_Bool);
IDENT(_Complex);
IDENT_RESERVED(_Imaginary);

/* C11 keywords */
IDENT(_Alignas);
IDENT_RESERVED(_Alignof);
IDENT_RESERVED(_Atomic);
IDENT_RESERVED(_Generic);
IDENT(_Noreturn);
IDENT_RESERVED(_Static_assert);
IDENT(_Thread_local);

/* Special case for L'\t' */
IDENT(L);

/* Extended gcc identifiers */
IDENT(asm); IDENT_RESERVED(__asm); IDENT_RESERVED(__asm__);
IDENT(alignof); IDENT_RESERVED(__alignof); IDENT_RESERVED(__alignof__);
IDENT_RESERVED(__sizeof_ptr__);
IDENT_RESERVED(__builtin_types_compatible_p);
IDENT_RESERVED(__builtin_offsetof);
IDENT_RESERVED(__label__);

/* Preprocessor idents.  Direct use of __IDENT avoids mentioning the keyword
 * itself by name, preventing these tokens from expanding when compiling
 * sparse. */
IDENT(defined);
```



```
IDENT(once);
__IDENT(pragma_ident, "__pragma__", 0);
__IDENT(_Pragma_ident, "_Pragma", 0);
__IDENT(__VA_ARGS___ident, "__VA_ARGS__", 0);
__IDENT(__LINE___ident, "__LINE__", 0);
__IDENT(__FILE___ident, "__FILE__", 0);
__IDENT(__DATE___ident, "__DATE__", 0);
__IDENT(__TIME___ident, "__TIME__", 0);
__IDENT(__func___ident, "__func__", 0);
__IDENT(__FUNCTION___ident, "__FUNCTION__", 0);
__IDENT(__PRETTY_FUNCTION___ident, "__PRETTY_FUNCTION__", 0);
__IDENT(__COUNTER___ident, "__COUNTER__", 0);

/* Sparse commands */
IDENT_RESERVED(__context__);
IDENT_RESERVED(__range__);

/* Magic function names we recognize */
IDENT(memset); IDENT(memcpy);
IDENT(copy_to_user); IDENT(copy_from_user);
IDENT(main);

#undef __IDENT
#undef IDENT
#undef IDENT_RESERVED
```

```
**********************************************************
   15695 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/inline.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Sparse - a semantic source parser.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include <stdlib.h>
#include <stdio.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"

static struct expression * dup_expression(struct expression *expr)
{
        struct expression *dup = alloc_expression(expr->pos, expr->type);
        *dup = *expr;
        return dup;
}

static struct statement * dup_statement(struct statement *stmt)
{
        struct statement *dup = alloc_statement(stmt->pos, stmt->type);
        *dup = *stmt;
        return dup;
}

static struct symbol *copy_symbol(struct position pos, struct symbol *sym)
{
        if (!sym)
                return sym;
        if (sym->ctype.modifiers & (MOD_STATIC | MOD_EXTERN | MOD_TOPLEVEL | MOD
                return sym;
        if (!sym->replace) {
                warning(pos, "unreplaced symbol '%s'", show_ident(sym->ident));
                return sym;
        }
        return sym->replace;
```



```
}

static struct symbol_list *copy_symbol_list(struct symbol_list *src)
{
        struct symbol_list *dst = NULL;
        struct symbol *sym;

        FOR_EACH_PTR(src, sym) {
                struct symbol *newsym = copy_symbol(sym->pos, sym);
                add_symbol(&dst, newsym);
        } END_FOR_EACH_PTR(sym);
        return dst;
}

static struct expression * copy_expression(struct expression *expr)
{
        if (!expr)
                return NULL;

        switch (expr->type) {
        /*
         * EXPR_SYMBOL is the interesting case, we may need to replace the
         * symbol to the new copy.
         */
        case EXPR_SYMBOL: {
                struct symbol *sym = copy_symbol(expr->pos, expr->symbol);
                if (sym == expr->symbol)
                        break;
                expr = dup_expression(expr);
                expr->symbol = sym;
                break;
        }

        /* Atomics, never change, just return the expression directly */
        case EXPR_VALUE:
        case EXPR_STRING:
        case EXPR_FVALUE:
        case EXPR_TYPE:
                break;

        /* Unops: check if the subexpression is unique */
        case EXPR_PREOP:
        case EXPR_POSTOP: {
                struct expression *unop = copy_expression(expr->unop);
                if (expr->unop == unop)
                        break;
                expr = dup_expression(expr);
                expr->unop = unop;
                break;
        }

        case EXPR_SLICE: {
                struct expression *base = copy_expression(expr->base);
                expr = dup_expression(expr);
                expr->base = base;
                break;
        }

        /* Binops: copy left/right expressions */
        case EXPR_BINOP:
        case EXPR_COMMA:
        case EXPR_COMPARE:
        case EXPR_LOGICAL: {
                struct expression *left = copy_expression(expr->left);
                struct expression *right = copy_expression(expr->right);
                if (left == expr->left && right == expr->right)
```

```
127                         break;
128                 expr = dup_expression(expr);
129                 expr->left = left;
130                 expr->right = right;
131                 break;
132         }

134         case EXPR_ASSIGNMENT: {
135                 struct expression *left = copy_expression(expr->left);
136                 struct expression *right = copy_expression(expr->right);
137                 if (expr->op == '=' && left == expr->left && right == expr->righ
138                         break;
139                 expr = dup_expression(expr);
140                 expr->left = left;
141                 expr->right = right;
142                 break;
143         }

145         /* Dereference */
146         case EXPR_DEREF: {
147                 struct expression *deref = copy_expression(expr->deref);
148                 expr = dup_expression(expr);
149                 expr->deref = deref;
150                 break;
151         }

153         /* Cast/sizeof/__alignof__ */
154         case EXPR_CAST:
155                 if (expr->cast_expression->type == EXPR_INITIALIZER) {
156                         struct expression *cast = expr->cast_expression;
157                         struct symbol *sym = expr->cast_type;
158                         expr = dup_expression(expr);
159                         expr->cast_expression = copy_expression(cast);
160                         expr->cast_type = alloc_symbol(sym->pos, sym->type);
161                         *expr->cast_type = *sym;
162                         break;
163                 }
164         case EXPR_FORCE_CAST:
165         case EXPR_IMPLIED_CAST:
166         case EXPR_SIZEOF:
167         case EXPR_PTRSIZEOF:
168         case EXPR_ALIGNOF: {
169                 struct expression *cast = copy_expression(expr->cast_expression)
170                 if (cast == expr->cast_expression)
171                         break;
172                 expr = dup_expression(expr);
173                 expr->cast_expression = cast;
174                 break;
175         }

177         /* Conditional expression */
178         case EXPR_SELECT:
179         case EXPR_CONDITIONAL: {
180                 struct expression *cond = copy_expression(expr->conditional);
181                 struct expression *true = copy_expression(expr->cond_true);
182                 struct expression *false = copy_expression(expr->cond_false);
183                 if (cond == expr->conditional && true == expr->cond_true && fals
184                         break;
185                 expr = dup_expression(expr);
186                 expr->conditional = cond;
187                 expr->cond_true = true;
188                 expr->cond_false = false;
189                 break;
190         }

192         /* Statement expression */
```



```
193         case EXPR_STATEMENT: {
194                 struct statement *stmt = alloc_statement(expr->pos, STMT_COMPOUN
195                 copy_statement(expr->statement, stmt);
196                 expr = dup_expression(expr);
197                 expr->statement = stmt;
198                 break;
199         }

201         /* Call expression */
202         case EXPR_CALL: {
203                 struct expression *fn = copy_expression(expr->fn);
204                 struct expression_list *list = expr->args;
205                 struct expression *arg;

207                 expr = dup_expression(expr);
208                 expr->fn = fn;
209                 expr->args = NULL;
210                 FOR_EACH_PTR(list, arg) {
211                         add_expression(&expr->args, copy_expression(arg));
212                 } END_FOR_EACH_PTR(arg);
213                 break;
214         }

216         /* Initializer list statement */
217         case EXPR_INITIALIZER: {
218                 struct expression_list *list = expr->expr_list;
219                 struct expression *entry;
220                 expr = dup_expression(expr);
221                 expr->expr_list = NULL;
222                 FOR_EACH_PTR(list, entry) {
223                         add_expression(&expr->expr_list, copy_expression(entry))
224                 } END_FOR_EACH_PTR(entry);
225                 break;
226         }

228         /* Label in inline function - hmm. */
229         case EXPR_LABEL: {
230                 struct symbol *label_symbol = copy_symbol(expr->pos, expr->label
231                 expr = dup_expression(expr);
232                 expr->label_symbol = label_symbol;
233                 break;
234         }

236         case EXPR_INDEX: {
237                 struct expression *sub_expr = copy_expression(expr->idx_expressi
238                 expr = dup_expression(expr);
239                 expr->idx_expression = sub_expr;
240                 break;
241         }
242
243         case EXPR_IDENTIFIER: {
244                 struct expression *sub_expr = copy_expression(expr->ident_expres
245                 expr = dup_expression(expr);
246                 expr->ident_expression = sub_expr;
247                 break;
248         }

250         /* Position in initializer.. */
251         case EXPR_POS: {
252                 struct expression *val = copy_expression(expr->init_expr);
253                 expr = dup_expression(expr);
254                 expr->init_expr = val;
255                 break;
256         }
257         case EXPR_OFFSETOF: {
258                 struct expression *val = copy_expression(expr->down);
```

```
259                 if (expr->op == '.') {
260                         if (expr->down != val) {
261                                 expr = dup_expression(expr);
262                                 expr->down = val;
263                         }
264                 } else {
265                         struct expression *idx = copy_expression(expr->index);
266                         if (expr->down != val || expr->index != idx) {
267                                 expr = dup_expression(expr);
268                                 expr->down = val;
269                                 expr->index = idx;
270                         }
271                 }
272                 break;
273         }
274         default:
275                 warning(expr->pos, "trying to copy expression type %d", expr->ty
276         }
277         return expr;
278 }

280 static struct expression_list *copy_asm_constraints(struct expression_list *in)
281 {
282         struct expression_list *out = NULL;
283         struct expression *expr;
284         int state = 0;

286         FOR_EACH_PTR(in, expr) {
287                 switch (state) {
288                 case 0: /* identifier */
289                 case 1: /* constraint */
290                         state++;
291                         add_expression(&out, expr);
292                         continue;
293                 case 2: /* expression */
294                         state = 0;
295                         add_expression(&out, copy_expression(expr));
296                         continue;
297                 }
298         } END_FOR_EACH_PTR(expr);
299         return out;
300 }

302 static void set_replace(struct symbol *old, struct symbol *new)
303 {
304         new->replace = old;
305         old->replace = new;
306 }

308 static void unset_replace(struct symbol *sym)
309 {
310         struct symbol *r = sym->replace;
311         if (!r) {
312                 warning(sym->pos, "symbol '%s' not replaced?", show_ident(sym->i
313                 return;
314         }
315         r->replace = NULL;
316         sym->replace = NULL;
317 }

319 static void unset_replace_list(struct symbol_list *list)
320 {
321         struct symbol *sym;
322         FOR_EACH_PTR(list, sym) {
323                 unset_replace(sym);
324         } END_FOR_EACH_PTR(sym);
```



```
325 }

327 static struct statement *copy_one_statement(struct statement *stmt)
328 {
329         if (!stmt)
330                 return NULL;
331         switch(stmt->type) {
332         case STMT_NONE:
333                 break;
334         case STMT_DECLARATION: {
335                 struct symbol *sym;
336                 struct statement *newstmt = dup_statement(stmt);
337                 newstmt->declaration = NULL;
338                 FOR_EACH_PTR(stmt->declaration, sym) {
339                         struct symbol *newsym = copy_symbol(stmt->pos, sym);
340                         if (newsym != sym)
341                                 newsym->initializer = copy_expression(sym->initi
342                         add_symbol(&newstmt->declaration, newsym);
343                 } END_FOR_EACH_PTR(sym);
344                 stmt = newstmt;
345                 break;
346         }
347         case STMT_CONTEXT:
348         case STMT_EXPRESSION: {
349                 struct expression *expr = copy_expression(stmt->expression);
350                 if (expr == stmt->expression)
351                         break;
352                 stmt = dup_statement(stmt);
353                 stmt->expression = expr;
354                 break;
355         }
356         case STMT_RANGE: {
357                 struct expression *expr = copy_expression(stmt->range_expression
358                 if (expr == stmt->expression)
359                         break;
360                 stmt = dup_statement(stmt);
361                 stmt->range_expression = expr;
362                 break;
363         }
364         case STMT_COMPOUND: {
365                 struct statement *new = alloc_statement(stmt->pos, STMT_COMPOUND
366                 copy_statement(stmt, new);
367                 stmt = new;
368                 break;
369         }
370         case STMT_IF: {
371                 struct expression *cond = stmt->if_conditional;
372                 struct statement *true = stmt->if_true;
373                 struct statement *false = stmt->if_false;

375                 cond = copy_expression(cond);
376                 true = copy_one_statement(true);
377                 false = copy_one_statement(false);
378                 if (stmt->if_conditional == cond &&
379                     stmt->if_true == true &&
380                     stmt->if_false == false)
381                         break;
382                 stmt = dup_statement(stmt);
383                 stmt->if_conditional = cond;
384                 stmt->if_true = true;
385                 stmt->if_false = false;
386                 break;
387         }
388         case STMT_RETURN: {
389                 struct expression *retval = copy_expression(stmt->ret_value);
390                 struct symbol *sym = copy_symbol(stmt->pos, stmt->ret_target);
```

```
392                 stmt = dup_statement(stmt);
393                 stmt->ret_value = retval;
394                 stmt->ret_target = sym;
395                 break;
396         }
397         case STMT_CASE: {
398                 stmt = dup_statement(stmt);
399                 stmt->case_label = copy_symbol(stmt->pos, stmt->case_label);
400                 stmt->case_label->stmt = stmt;
401                 stmt->case_expression = copy_expression(stmt->case_expression);
402                 stmt->case_to = copy_expression(stmt->case_to);
403                 stmt->case_statement = copy_one_statement(stmt->case_statement);
404                 break;
405         }
406         case STMT_SWITCH: {
407                 struct symbol *switch_break = copy_symbol(stmt->pos, stmt->switc
408                 struct symbol *switch_case = copy_symbol(stmt->pos, stmt->switch
409                 struct expression *expr = copy_expression(stmt->switch_expressio
410                 struct statement *switch_stmt = copy_one_statement(stmt->switch_

412                 stmt = dup_statement(stmt);
413                 switch_case->symbol_list = copy_symbol_list(switch_case->symbol_
414                 stmt->switch_break = switch_break;
415                 stmt->switch_case = switch_case;
416                 stmt->switch_expression = expr;
417                 stmt->switch_statement = switch_stmt;
418                 break;
419         }
420         case STMT_ITERATOR: {
421                 stmt = dup_statement(stmt);
422                 stmt->iterator_break = copy_symbol(stmt->pos, stmt->iterator_bre
423                 stmt->iterator_continue = copy_symbol(stmt->pos, stmt->iterator_
424                 stmt->iterator_syms = copy_symbol_list(stmt->iterator_syms);

426                 stmt->iterator_pre_statement = copy_one_statement(stmt->iterator
427                 stmt->iterator_pre_condition = copy_expression(stmt->iterator_pr

429                 stmt->iterator_statement = copy_one_statement(stmt->iterator_sta

431                 stmt->iterator_post_statement = copy_one_statement(stmt->iterato
432                 stmt->iterator_post_condition = copy_expression(stmt->iterator_p
433                 break;
434         }
435         case STMT_LABEL: {
436                 stmt = dup_statement(stmt);
437                 stmt->label_identifier = copy_symbol(stmt->pos, stmt->label_iden
438                 stmt->label_statement = copy_one_statement(stmt->label_statement
439                 break;
440         }
441         case STMT_GOTO: {
442                 stmt = dup_statement(stmt);
443                 stmt->goto_label = copy_symbol(stmt->pos, stmt->goto_label);
444                 stmt->goto_expression = copy_expression(stmt->goto_expression);
445                 stmt->target_list = copy_symbol_list(stmt->target_list);
446                 break;
447         }
448         case STMT_ASM: {
449                 stmt = dup_statement(stmt);
450                 stmt->asm_inputs = copy_asm_constraints(stmt->asm_inputs);
451                 stmt->asm_outputs = copy_asm_constraints(stmt->asm_outputs);
452                 /* no need to dup "clobbers", since they are all constant string
453                 break;
454         }
455         default:
456                 warning(stmt->pos, "trying to copy statement type %d", stmt->typ
```



```
457                 break;
458         }
459         return stmt;
460 }

462 /*
463  * Copy a statement tree from 'src' to 'dst', where both
464  * source and destination are of type STMT_COMPOUND.
465  *
466  * We do this for the tree-level inliner.
467  *
468  * This doesn't do the symbol replacement right: it's not
469  * re-entrant.
470  */
471 void copy_statement(struct statement *src, struct statement *dst)
472 {
473         struct statement *stmt;

475         FOR_EACH_PTR(src->stmts, stmt) {
476                 add_statement(&dst->stmts, copy_one_statement(stmt));
477         } END_FOR_EACH_PTR(stmt);
478         dst->args = copy_one_statement(src->args);
479         dst->ret = copy_symbol(src->pos, src->ret);
480         dst->inline_fn = src->inline_fn;
481 }

483 static struct symbol *create_copy_symbol(struct symbol *orig)
484 {
485         struct symbol *sym = orig;
486         if (orig) {
487                 sym = alloc_symbol(orig->pos, orig->type);
488                 *sym = *orig;
489                 sym->bb_target = NULL;
490                 sym->pseudo = NULL;
491                 set_replace(orig, sym);
492                 orig = sym;
493         }
494         return orig;
495 }

497 static struct symbol_list *create_symbol_list(struct symbol_list *src)
498 {
499         struct symbol_list *dst = NULL;
500         struct symbol *sym;

502         FOR_EACH_PTR(src, sym) {
503                 struct symbol *newsym = create_copy_symbol(sym);
504                 add_symbol(&dst, newsym);
505         } END_FOR_EACH_PTR(sym);
506         return dst;
507 }

509 int inline_function(struct expression *expr, struct symbol *sym)
510 {
511         struct symbol_list * fn_symbol_list;
512         struct symbol *fn = sym->ctype.base_type;
513         struct expression_list *arg_list = expr->args;
514         struct statement *stmt = alloc_statement(expr->pos, STMT_COMPOUND);
515         struct symbol_list *name_list, *arg_decl;
516         struct symbol *name;
517         struct expression *arg;

519         if (!fn->inline_stmt) {
520                 sparse_error(fn->pos, "marked inline, but without a definition")
521                 return 0;
522         }
```

```
523         if (fn->expanding)
524                 return 0;

526         fn->expanding = 1;

528         name_list = fn->arguments;

530         expr->type = EXPR_STATEMENT;
531         expr->statement = stmt;
532         expr->ctype = fn->ctype.base_type;

534         fn_symbol_list = create_symbol_list(sym->inline_symbol_list);

536         arg_decl = NULL;
537         PREPARE_PTR_LIST(name_list, name);
538         FOR_EACH_PTR(arg_list, arg) {
539                 struct symbol *a = alloc_symbol(arg->pos, SYM_NODE);

541                 a->ctype.base_type = arg->ctype;
542                 if (name) {
543                         *a = *name;
544                         set_replace(name, a);
545                         add_symbol(&fn_symbol_list, a);
546                 }
547                 a->initializer = arg;
548                 add_symbol(&arg_decl, a);

550                 NEXT_PTR_LIST(name);
551         } END_FOR_EACH_PTR(arg);
552         FINISH_PTR_LIST(name);

554         copy_statement(fn->inline_stmt, stmt);

556         if (arg_decl) {
557                 struct statement *decl = alloc_statement(expr->pos, STMT_DECLARA
558                 decl->declaration = arg_decl;
559                 stmt->args = decl;
560         }
561         stmt->inline_fn = sym;

563         unset_replace_list(fn_symbol_list);

565         evaluate_statement(stmt);

567         fn->expanding = 0;
568         return 1;
569 }

571 void uninline(struct symbol *sym)
572 {
573         struct symbol *fn = sym->ctype.base_type;
574         struct symbol_list *arg_list = fn->arguments;
575         struct symbol *p;

577         sym->symbol_list = create_symbol_list(sym->inline_symbol_list);
578         FOR_EACH_PTR(arg_list, p) {
579                 p->replace = p;
580         } END_FOR_EACH_PTR(p);
581         fn->stmt = alloc_statement(fn->pos, STMT_COMPOUND);
582         copy_statement(fn->inline_stmt, fn->stmt);
583         unset_replace_list(sym->symbol_list);
584         unset_replace_list(arg_list);
585 }
```

```
**********************************************************
   42676 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/lib.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * 'sparse' library helper routines.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <ctype.h>
#include <fcntl.h>
#include <stdarg.h>
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <assert.h>

#include <sys/types.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"
#include "scope.h"
#include "linearize.h"
#include "target.h"
#include "version.h"

static const char *progname;

int sparse_errors = 0;
int sparse_warnings = 0;

int verbose, optimize, optimize_size, preprocessing;
int die_if_error = 0;
int parse_error;
int has_error = 0;

#ifndef __GNUC__
# define __GNUC__ 2
# define __GNUC_MINOR__ 95
```



```
# define __GNUC_PATCHLEVEL__ 0
#endif

int gcc_major = __GNUC__;
int gcc_minor = __GNUC_MINOR__;
int gcc_patchlevel = __GNUC_PATCHLEVEL__;

static const char *gcc_base_dir = GCC_BASE;
static const char *multiarch_dir = MULTIARCH_TRIPLET;

struct token *skip_to(struct token *token, int op)
{
        while (!match_op(token, op) && !eof_token(token))
                token = token->next;
        return token;
}

struct token *expect(struct token *token, int op, const char *where)
{
        if (!match_op(token, op)) {
                static struct token bad_token;
                if (token != &bad_token) {
                        bad_token.next = token;
                        sparse_error(token->pos, "Expected %s %s", show_special(
                        sparse_error(token->pos, "got %s", show_token(token));
                }
                if (op == ';')
                        return skip_to(token, op);
                return &bad_token;
        }
        return token->next;
}

unsigned int hexval(unsigned int c)
{
        int retval = 256;
        switch (c) {
        case '0'...'9':
                retval = c - '0';
                break;
        case 'a'...'f':
                retval = c - 'a' + 10;
                break;
        case 'A'...'F':
                retval = c - 'A' + 10;
                break;
        }
        return retval;
}

static void do_warn(const char *type, struct position pos, const char * fmt, va_
{
        static char buffer[512];
        const char *name;

        vsprintf(buffer, fmt, args);
        name = stream_name(pos.stream);

        fprintf(stderr, "%s: %s:%d:%d: %s%s\n",
                progname, name, pos.line, pos.pos, type, buffer);
}

static int max_warnings = 100;
static int show_info = 1;

void info(struct position pos, const char * fmt, ...)
```

```
{
        va_list args;

        if (!show_info)
                return;
        va_start(args, fmt);
        do_warn("", pos, fmt, args);
        va_end(args);
}

static void do_error(struct position pos, const char * fmt, va_list args)
{
        static int errors = 0;

        parse_error = 1;
        die_if_error = 1;
        show_info = 1;
        /* Shut up warnings after an error */
        has_error |= ERROR_CURR_PHASE;
        if (errors > 100) {
                static int once = 0;
                show_info = 0;
                if (once)
                        return;
                fmt = "too many errors";
                once = 1;
        }

        do_warn("error: ", pos, fmt, args);
        errors++;
}

void warning(struct position pos, const char * fmt, ...)
{
        va_list args;

        if (Wsparse_error) {
                va_start(args, fmt);
                do_error(pos, fmt, args);
                va_end(args);
                return;
        }

        if (!max_warnings || has_error) {
                show_info = 0;
                return;
        }

        if (!--max_warnings) {
                show_info = 0;
                fmt = "too many warnings";
        }

        va_start(args, fmt);
        do_warn("warning: ", pos, fmt, args);
        va_end(args);
}

void sparse_error(struct position pos, const char * fmt, ...)
{
        va_list args;
        va_start(args, fmt);
        do_error(pos, fmt, args);
        va_end(args);
}
```



```
void expression_error(struct expression *expr, const char *fmt, ...)
{
        va_list args;
        va_start(args, fmt);
        do_error(expr->pos, fmt, args);
        va_end(args);
        expr->ctype = &bad_ctype;
}

NORETURN_ATTR
void error_die(struct position pos, const char * fmt, ...)
{
        va_list args;
        va_start(args, fmt);
        do_warn("error: ", pos, fmt, args);
        va_end(args);
        exit(1);
}

NORETURN_ATTR
void die(const char *fmt, ...)
{
        va_list args;
        static char buffer[512];

        va_start(args, fmt);
        vsnprintf(buffer, sizeof(buffer), fmt, args);
        va_end(args);

        fprintf(stderr, "%s: %s\n", progname, buffer);
        exit(1);
}

static struct token *pre_buffer_begin = NULL;
static struct token *pre_buffer_end = NULL;

int Waddress = 0;
int Waddress_space = 1;
int Wbitwise = 1;
int Wcast_to_as = 0;
int Wcast_truncate = 1;
int Wconstant_suffix = 0;
int Wconstexpr_not_const = 0;
int Wcontext = 1;
int Wdecl = 1;
int Wdeclarationafterstatement = -1;
int Wdefault_bitfield_sign = 0;
int Wdesignated_init = 1;
int Wdo_while = 0;
int Wimplicit_int = 1;
int Winit_cstring = 0;
int Wenum_mismatch = 1;
int Wexternal_function_has_definition = 1;
int Wsparse_error = 0;
int Wmemcpy_max_count = 1;
int Wnon_pointer_null = 1;
int Wold_initializer = 1;
int Wold_style_definition = 1;
int Wone_bit_signed_bitfield = 1;
int Woverride_init = 1;
int Woverride_init_all = 0;
int Woverride_init_whole_range = 0;
int Wparen_string = 0;
int Wpointer_arith = 0;
int Wptr_subtraction_blows = 0;
int Wreturn_void = 0;
```

```
int Wshadow = 0;
int Wsizeof_bool = 0;
int Wstrict_prototypes = 1;
int Wtautological_compare = 0;
int Wtransparent_union = 0;
int Wtypesign = 0;
int Wundef = 0;
int Wuninitialized = 1;
int Wunknown_attribute = 0;
int Wvla = 1;

int dump_macro_defs = 0;

int dbg_entry = 0;
int dbg_dead = 0;

int fmem_report = 0;
int fdump_linearize;
unsigned long long fmemcpy_max_count = 100000;

int preprocess_only;

static enum { STANDARD_C89,
              STANDARD_C94,
              STANDARD_C99,
              STANDARD_C11,
              STANDARD_GNU11,
              STANDARD_GNU89,
              STANDARD_GNU99, } standard = STANDARD_GNU89;

#define ARCH_LP32  0
#define ARCH_LP64  1
#define ARCH_LLP64 2

#ifdef __x86_64__
#define ARCH_M64_DEFAULT ARCH_LP64
#else
#define ARCH_M64_DEFAULT ARCH_LP32
#endif

int arch_m64 = ARCH_M64_DEFAULT;
int arch_msize_long = 0;

#ifdef __BIG_ENDIAN__
#define ARCH_BIG_ENDIAN 1
#else
#define ARCH_BIG_ENDIAN 0
#endif
int arch_big_endian = ARCH_BIG_ENDIAN;


#define CMDLINE_INCLUDE 20
static int cmdline_include_nr = 0;
static char *cmdline_include[CMDLINE_INCLUDE];


void add_pre_buffer(const char *fmt, ...)
{
        va_list args;
        unsigned int size;
        struct token *begin, *end;
        char buffer[4096];

        va_start(args, fmt);
        size = vsnprintf(buffer, sizeof(buffer), fmt, args);
        va_end(args);
```



```
        begin = tokenize_buffer(buffer, size, &end);
        if (!pre_buffer_begin)
                pre_buffer_begin = begin;
        if (pre_buffer_end)
                pre_buffer_end->next = begin;
        pre_buffer_end = end;
}

static char **handle_switch_D(char *arg, char **next)
{
        const char *name = arg + 1;
        const char *value = "1";

        if (!*name) {
                arg = *++next;
                if (!arg)
                        die("argument to ‘-D’ is missing");
                name = arg;
        }

        for (;;arg++) {
                char c;
                c = *arg;
                if (!c)
                        break;
                if (c == ’=’) {
                        *arg = ’\0’;
                        value = arg + 1;
                        break;
                }
        }
        add_pre_buffer("#define %s %s\n", name, value);
        return next;
}

static char **handle_switch_E(char *arg, char **next)
{
        if (arg[1] == ’\0’)
                preprocess_only = 1;
        return next;
}

static char **handle_switch_I(char *arg, char **next)
{
        char *path = arg+1;

        switch (arg[1]) {
        case ’-’:
                add_pre_buffer("#split_include\n");
                break;

        case ’\0’:      /* Plain "-I" */
                path = *++next;
                if (!path)
                        die("missing argument for -I option");
                /* Fall through */
        default:
                add_pre_buffer("#add_include \"%s/\"\n", path);
        }
        return next;
}

static void add_cmdline_include(char *filename)
{
        if (cmdline_include_nr >= CMDLINE_INCLUDE)
                die("too many include files for %s\n", filename);
```

```
391         cmdline_include[cmdline_include_nr++] = filename;
392 }

394 static char **handle_switch_i(char *arg, char **next)
395 {
396         if (*next && !strcmp(arg, "include"))
397                 add_cmdline_include(*++next);
398         else if (*next && !strcmp(arg, "imacros"))
399                 add_cmdline_include(*++next);
400         else if (*next && !strcmp(arg, "isystem")) {
401                 char *path = *++next;
402                 if (!path)
403                         die("missing argument for -isystem option");
404                 add_pre_buffer("#add_isystem \"%s/\"\n", path);
405         } else if (*next && !strcmp(arg, "idirafter")) {
406                 char *path = *++next;
407                 if (!path)
408                         die("missing argument for -idirafter option");
409                 add_pre_buffer("#add_dirafter \"%s/\"\n", path);
410         }
411         return next;
412 }

414 static char **handle_switch_M(char *arg, char **next)
415 {
416         if (!strcmp(arg, "MF") || !strcmp(arg,"MQ") || !strcmp(arg,"MT")) {
417                 if (!*next)
418                         die("missing argument for -%s option", arg);
419                 return next + 1;
420         }
421         return next;
422 }

424 static char **handle_multiarch_dir(char *arg, char **next)
425 {
426         multiarch_dir = *++next;
427         if (!multiarch_dir)
428                 die("missing argument for -multiarch-dir option");
429         return next;
430 }

432 static char **handle_switch_m(char *arg, char **next)
433 {
434         if (!strcmp(arg, "m64")) {
435                 arch_m64 = ARCH_LP64;
436         } else if (!strcmp(arg, "m32")) {
437                 arch_m64 = ARCH_LP32;
438         } else if (!strcmp(arg, "msize-llp64")) {
439                 arch_m64 = ARCH_LLP64;
440         } else if (!strcmp(arg, "msize-long")) {
441                 arch_msize_long = 1;
442         } else if (!strcmp(arg, "multiarch-dir")) {
443                 return handle_multiarch_dir(arg, next);
444         } else if (!strcmp(arg, "mbig-endian")) {
445                 arch_big_endian = 1;
446         } else if (!strcmp(arg, "mlittle-endian")) {
447                 arch_big_endian = 0;
448         }
449         return next;
450 }

452 static void handle_arch_m64_finalize(void)
453 {
454         switch (arch_m64) {
455         case ARCH_LP32:
456                 /* default values */
```



```
457 #if defined(__x86_64__) || defined (__i386)
458                 add_pre_buffer("#weak_define __i386__ 1\n");
459                 add_pre_buffer("#weak_define __i386 1\n");
460                 add_pre_buffer("#weak_define i386 1\n");
461 #endif
462                 return;
463         case ARCH_LP64:
464                 bits_in_long = 64;
465                 max_int_alignment = 8;
466                 size_t_ctype = &ulong_ctype;
467                 ssize_t_ctype = &long_ctype;
468                 add_pre_buffer("#weak_define __LP64__ 1\n");
469                 add_pre_buffer("#weak_define __LP64 1\n");
470                 add_pre_buffer("#weak_define _LP64 1\n");
471                 goto case_64bit_common;
472         case ARCH_LLP64:
473                 bits_in_long = 32;
474                 max_int_alignment = 4;
475                 size_t_ctype = &ullong_ctype;
476                 ssize_t_ctype = &llong_ctype;
477                 add_pre_buffer("#weak_define __LLP64__ 1\n");
478                 goto case_64bit_common;
479         case_64bit_common:
480                 bits_in_pointer = 64;
481                 pointer_alignment = 8;
482 #if defined(__x86_64__) || defined (__i386)
483                 add_pre_buffer("#weak_define __x86_64__ 1\n");
484                 add_pre_buffer("#weak_define __x86_64 1\n");
485 #endif
486                 break;
487         }
488 }

490 static void handle_arch_msize_long_finalize(void)
491 {
492         if (arch_msize_long) {
493                 size_t_ctype = &ulong_ctype;
494                 ssize_t_ctype = &long_ctype;
495         }
496 }

498 static void handle_arch_finalize(void)
499 {
500         handle_arch_m64_finalize();
501         handle_arch_msize_long_finalize();
502 }


505 static int handle_simple_switch(const char *arg, const char *name, int *flag)
506 {
507         int val = 1;

509         // Prefixe "no-" mean to turn flag off.
510         if (strncmp(arg, "no-", 3) == 0) {
511                 arg += 3;
512                 val = 0;
513         }

515         if (strcmp(arg, name) == 0) {
516                 *flag = val;
517                 return 1;
518         }

520         // not handled
521         return 0;
522 }
```

```
524 static char **handle_switch_o(char *arg, char **next)
525 {
526         if (!strcmp (arg, "o")) {       // "-o foo"
527                 if (!*++next)
528                         die("argument to '-o' is missing");
529         }
530         // else "-ofoo"

532         return next;
533 }

535 static const struct warning {
536         const char *name;
537         int *flag;
538 } warnings[] = {
539         { "address", &Waddress },
540         { "address-space", &Waddress_space },
541         { "bitwise", &Wbitwise },
542         { "cast-to-as", &Wcast_to_as },
543         { "cast-truncate", &Wcast_truncate },
544         { "constant-suffix", &Wconstant_suffix },
545         { "constexpr-not-const", &Wconstexpr_not_const},
546         { "context", &Wcontext },
547         { "decl", &Wdecl },
548         { "declaration-after-statement", &Wdeclarationafterstatement },
549         { "default-bitfield-sign", &Wdefault_bitfield_sign },
550         { "designated-init", &Wdesignated_init },
551         { "do-while", &Wdo_while },
552         { "enum-mismatch", &Wenum_mismatch },
553         { "external-function-has-definition", &Wexternal_function_has_definition
554         { "implicit-int", &Wimplicit_int },
555         { "init-cstring", &Winit_cstring },
556         { "memcpy-max-count", &Wmemcpy_max_count },
557         { "non-pointer-null", &Wnon_pointer_null },
558         { "old-initializer", &Wold_initializer },
559         { "old-style-definition", &Wold_style_definition },
560         { "one-bit-signed-bitfield", &Wone_bit_signed_bitfield },
561         { "override-init", &Woverride_init },
562         { "override-init-all", &Woverride_init_all },
563         { "paren-string", &Wparen_string },
564         { "ptr-subtraction-blows", &Wptr_subtraction_blows },
565         { "return-void", &Wreturn_void },
566         { "shadow", &Wshadow },
567         { "sizeof-bool", &Wsizeof_bool },
568         { "strict-prototypes", &Wstrict_prototypes },
569         { "pointer-arith", &Wpointer_arith },
570         { "sparse-error", &Wsparse_error },
571         { "tautological-compare", &Wtautological_compare },
572         { "transparent-union", &Wtransparent_union },
573         { "typesign", &Wtypesign },
574         { "undef", &Wundef },
575         { "uninitialized", &Wuninitialized },
576         { "unknown-attribute", &Wunknown_attribute },
577         { "vla", &Wvla },
578 };

580 enum {
581         WARNING_OFF,
582         WARNING_ON,
583         WARNING_FORCE_OFF
584 };


587 static char **handle_onoff_switch(char *arg, char **next, const struct warning w
588 {
```



```
589         int flag = WARNING_ON;
590         char *p = arg + 1;
591         unsigned i;

593         if (!strcmp(p, "sparse-all")) {
594                 for (i = 0; i < n; i++) {
595                         if (*warnings[i].flag != WARNING_FORCE_OFF && warnings[i
596                                 *warnings[i].flag = WARNING_ON;
597                 }
598         }

600         // Prefixes "no" and "no-" mean to turn warning off.
601         if (p[0] == 'n' && p[1] == 'o') {
602                 p += 2;
603                 if (p[0] == '-')
604                         p++;
605                 flag = WARNING_FORCE_OFF;
606         }

608         for (i = 0; i < n; i++) {
609                 if (!strcmp(p,warnings[i].name)) {
610                         *warnings[i].flag = flag;
611                         return next;
612                 }
613         }

615         // Unknown.
616         return NULL;
617 }

619 static char **handle_switch_W(char *arg, char **next)
620 {
621         char ** ret = handle_onoff_switch(arg, next, warnings, ARRAY_SIZE(warnin
622         if (ret)
623                 return ret;

625         // Unknown.
626         return next;
627 }

629 static struct warning debugs[] = {
630         { "entry", &dbg_entry},
631         { "dead", &dbg_dead},
632 };


635 static char **handle_switch_v(char *arg, char **next)
636 {
637         char ** ret = handle_onoff_switch(arg, next, debugs, ARRAY_SIZE(debugs))
638         if (ret)
639                 return ret;

641         // Unknown.
642         do {
643                 verbose++;
644         } while (*++arg == 'v');
645         return next;
646 }

648 static struct warning dumps[] = {
649         { "D", &dump_macro_defs},
650 };

652 static char **handle_switch_d(char *arg, char **next)
653 {
654         char ** ret = handle_onoff_switch(arg, next, dumps, ARRAY_SIZE(dumps));
```

```
655         if (ret)
656                 return ret;
657
658         return next;
659 }


662 static void handle_onoff_switch_finalize(const struct warning warnings[], int n)
663 {
664         unsigned i;
665
666         for (i = 0; i < n; i++) {
667                 if (*warnings[i].flag == WARNING_FORCE_OFF)
668                         *warnings[i].flag = WARNING_OFF;
669         }
670 }
671
672 static void handle_switch_W_finalize(void)
673 {
674         handle_onoff_switch_finalize(warnings, ARRAY_SIZE(warnings));
675
676         /* default Wdeclarationafterstatement based on the C dialect */
677         if (-1 == Wdeclarationafterstatement)
678         {
679                 switch (standard)
680                 {
681                         case STANDARD_C89:
682                         case STANDARD_C94:
683                                 Wdeclarationafterstatement = 1;
684                                 break;
685
686                         case STANDARD_C99:
687                         case STANDARD_GNU89:
688                         case STANDARD_GNU99:
689                         case STANDARD_C11:
690                         case STANDARD_GNU11:
691                                 Wdeclarationafterstatement = 0;
692                                 break;
693
694                         default:
695                                 assert (0);
696                 }
697
698         }
699 }
700
701 static void handle_switch_v_finalize(void)
702 {
703         handle_onoff_switch_finalize(debugs, ARRAY_SIZE(debugs));
704 }
705
706 static char **handle_switch_U(char *arg, char **next)
707 {
708         const char *name = arg + 1;
709         if (*name == '\0')
710                 name = *++next;
711         add_pre_buffer ("#undef %s\n", name);
712         return next;
713 }
714
715 static char **handle_switch_O(char *arg, char **next)
716 {
717         int level = 1;
718         if (arg[1] >= '0' && arg[1] <= '9')
719                 level = arg[1] - '0';
720         optimize = level;
```



```
721         optimize_size = arg[1] == 's';
722         return next;
723 }
724
725 static char **handle_switch_fmemcpy_max_count(char *arg, char **next)
726 {
727         unsigned long long val;
728         char *end;
729
730         val = strtoull(arg, &end, 0);
731         if (*end != '\0' || end == arg)
732                 die("error: missing argument to \"-fmemcpy-max-count=\"");
733
734         if (val == 0)
735                 val = ~0ULL;
736         fmemcpy_max_count = val;
737         return next;
738 }
739
740 static char **handle_switch_ftabstop(char *arg, char **next)
741 {
742         char *end;
743         unsigned long val;
744
745         if (*arg == '\0')
746                 die("error: missing argument to \"-ftabstop=\"");
747
748         /* we silently ignore silly values */
749         val = strtoul(arg, &end, 10);
750         if (*end == '\0' && 1 <= val && val <= 100)
751                 tabstop = val;
752
753         return next;
754 }
755
756 static int funsigned_char;
757 static void handle_funsigned_char(void)
758 {
759         if (funsigned_char) {
760                 char_ctype.ctype.modifiers &= ~MOD_SIGNED;
761                 char_ctype.ctype.modifiers |= MOD_UNSIGNED;
762         }
763 }
764
765         static char **handle_switch_fdump(char *arg, char **next)
766 {
767         if (!strncmp(arg, "linearize", 9)) {
768                 arg += 9;
769                 if (*arg == '\0')
770                         fdump_linearize = 1;
771                 else if (!strcmp(arg, "=only"))
772                         fdump_linearize = 2;
773                 else
774                         goto err;
775         }
776
777         /* ignore others flags */
778         return next;
779
780 err:
781         die("error: unknown flag \"-fdump-%s\"", arg);
782 }
783
784 static char **handle_switch_f(char *arg, char **next)
785 {
786         arg++;
```

```
788         if (!strncmp(arg, "tabstop=", 8))
789                 return handle_switch_ftabstop(arg+8, next);
790         if (!strncmp(arg, "dump-", 5))
791                 return handle_switch_fdump(arg+5, next);
792         if (!strncmp(arg, "memcpy-max-count=", 17))
793                 return handle_switch_fmemcpy_max_count(arg+17, next);

795         if (!strcmp(arg, "unsigned-char")) {
796                 funsigned_char = 1;
797                 return next;
798         }

800         /* handle switches w/ arguments above, boolean and only boolean below */
801         if (handle_simple_switch(arg, "mem-report", &fmem_report))
802                 return next;

804         return next;
805 }

807 static char **handle_switch_G(char *arg, char **next)
808 {
809         if (!strcmp (arg, "G") && *next)
810                 return next + 1; // "-G 0"
811         else
812                 return next;     // "-G0" or (bogus) terminal "-G"
813 }

815 static char **handle_switch_a(char *arg, char **next)
816 {
817         if (!strcmp (arg, "ansi"))
818                 standard = STANDARD_C89;

820         return next;
821 }

823 static char **handle_switch_s(char *arg, char **next)
824 {
825         if (!strncmp (arg, "std=", 4))
826         {
827                 arg += 4;

829                 if (!strcmp (arg, "c89") ||
830                     !strcmp (arg, "iso9899:1990"))
831                         standard = STANDARD_C89;

833                 else if (!strcmp (arg, "iso9899:199409"))
834                         standard = STANDARD_C94;

836                 else if (!strcmp (arg, "c99") ||
837                          !strcmp (arg, "c9x") ||
838                          !strcmp (arg, "iso9899:1999") ||
839                          !strcmp (arg, "iso9899:199x"))
840                         standard = STANDARD_C99;

842                 else if (!strcmp (arg, "gnu89"))
843                         standard = STANDARD_GNU89;

845                 else if (!strcmp (arg, "gnu99") || !strcmp (arg, "gnu9x"))
846                         standard = STANDARD_GNU99;

848                 else if (!strcmp(arg, "c11") ||
849                          !strcmp(arg, "c1x") ||
850                          !strcmp(arg, "iso9899:2011"))
851                         standard = STANDARD_C11;
```



```
853                 else if (!strcmp(arg, "gnu11"))
854                         standard = STANDARD_GNU11;

856                 else
857                         die ("Unsupported C dialect");
858         }

860         return next;
861 }

863 static char **handle_nostdinc(char *arg, char **next)
864 {
865         add_pre_buffer("#nostdinc\n");
866         return next;
867 }

869 static char **handle_switch_n(char *arg, char **next)
870 {
871         if (!strcmp (arg, "nostdinc"))
872                 return handle_nostdinc(arg, next);

874         return next;
875 }

877 static char **handle_base_dir(char *arg, char **next)
878 {
879         gcc_base_dir = *++next;
880         if (!gcc_base_dir)
881                 die("missing argument for -gcc-base-dir option");
882         return next;
883 }

885 static char **handle_no_lineno(char *arg, char **next)
886 {
887         no_lineno = 1;
888         return next;
889 }

891 static char **handle_switch_g(char *arg, char **next)
892 {
893         if (!strcmp (arg, "gcc-base-dir"))
894                 return handle_base_dir(arg, next);

896         return next;
897 }

899 static char **handle_version(char *arg, char **next)
900 {
901         printf("%s\n", SPARSE_VERSION);
902         exit(0);
903 }

905 static char **handle_param(char *arg, char **next)
906 {
907         char *value = NULL;

909         /* Ignore smatch's --param-mapper */
910         if (strcmp(arg, "-mapper") == 0)
911                 return next;


914         /* For now just skip any '--param=*' or '--param *' */
915         if (*arg == '\0') {
916                 value = *++next;
917         } else if (isspace((unsigned char)*arg) || *arg == '=') {
918                 value = ++arg;
```

```
919         }

921         if (!value)
922                 die("missing argument for --param option");

924         return next;
925 }

927 struct switches {
928         const char *name;
929         char **(*fn)(char *, char **);
930         unsigned int prefix:1;
931 };

933 static char **handle_long_options(char *arg, char **next)
934 {
935         static struct switches cmd[] = {
936                 { "param", handle_param, 1 },
937                 { "version", handle_version },
938                 { "nostdinc", handle_nostdinc },
939                 { "gcc-base-dir", handle_base_dir},
940                 { "no-lineno", handle_no_lineno},
941                 { NULL, NULL }
942         };
943         struct switches *s = cmd;

945         while (s->name) {
946                 int optlen = strlen(s->name);
947                 if (!strncmp(s->name, arg, optlen + !s->prefix))
948                         return s->fn(arg + optlen, next);
949                 s++;
950         }
951         return next;
952 }

954 static char **handle_switch(char *arg, char **next)
955 {
956         switch (*arg) {
957         case 'a': return handle_switch_a(arg, next);
958         case 'D': return handle_switch_D(arg, next);
959         case 'd': return handle_switch_d(arg, next);
960         case 'E': return handle_switch_E(arg, next);
961         case 'f': return handle_switch_f(arg, next);
962         case 'g': return handle_switch_g(arg, next);
963         case 'G': return handle_switch_G(arg, next);
964         case 'I': return handle_switch_I(arg, next);
965         case 'i': return handle_switch_i(arg, next);
966         case 'M': return handle_switch_M(arg, next);
967         case 'm': return handle_switch_m(arg, next);
968         case 'n': return handle_switch_n(arg, next);
969         case 'o': return handle_switch_o(arg, next);
970         case 'O': return handle_switch_O(arg, next);
971         case 's': return handle_switch_s(arg, next);
972         case 'U': return handle_switch_U(arg, next);
973         case 'v': return handle_switch_v(arg, next);
974         case 'W': return handle_switch_W(arg, next);
975         case '-': return handle_long_options(arg + 1, next);
976         default:
977                 break;
978         }

980         /*
981          * Ignore unknown command line options:
982          * they're probably gcc switches
983          */
984         return next;
```



```
985 }

987 static void predefined_sizeof(const char *name, unsigned bits)
988 {
989         add_pre_buffer("#weak_define __SIZEOF_%s__ %d\n", name, bits/8);
990 }

992 static void predefined_max(const char *name, const char *suffix, unsigned bits)
993 {
994         unsigned long long max = (1ULL << (bits - 1 )) - 1;

996         add_pre_buffer("#weak_define __%s_MAX__ %#llx%s\n", name, max, suffix);
997 }

999 static void predefined_type_size(const char *name, const char *suffix, unsigned
1000 {
1001         predefined_max(name, suffix, bits);
1002         predefined_sizeof(name, bits);
1003 }

1005 static void predefined_macros(void)
1006 {
1007         add_pre_buffer("#define __CHECKER__ 1\n");

1009         predefined_sizeof("SHORT", bits_in_short);
1010         predefined_max("SHRT", "", bits_in_short);
1011         predefined_max("SCHAR", "", bits_in_char);
1012         predefined_max("WCHAR", "", bits_in_wchar);
1013         add_pre_buffer("#weak_define __CHAR_BIT__ %d\n", bits_in_char);

1015         predefined_type_size("INT", "", bits_in_int);
1016         predefined_type_size("LONG", "L", bits_in_long);
1017         predefined_type_size("LONG_LONG", "LL", bits_in_longlong);

1019         predefined_sizeof("INT128", 128);

1021         predefined_sizeof("SIZE_T", bits_in_pointer);
1022         predefined_sizeof("PTRDIFF_T", bits_in_pointer);
1023         predefined_sizeof("POINTER", bits_in_pointer);

1025         predefined_sizeof("FLOAT", bits_in_float);
1026         predefined_sizeof("DOUBLE", bits_in_double);
1027         predefined_sizeof("LONG_DOUBLE", bits_in_longdouble);

1029         add_pre_buffer("#weak_define __%s_ENDIAN__ 1\n",
1030                 arch_big_endian ? "BIG" : "LITTLE");

1032         add_pre_buffer("#weak_define __ORDER_LITTLE_ENDIAN__ 1234\n");
1033         add_pre_buffer("#weak_define __ORDER_BIG_ENDIAN__ 4321\n");
1034         add_pre_buffer("#weak_define __ORDER_PDP_ENDIAN__ 3412\n");
1035         add_pre_buffer("#weak_define __BYTE_ORDER__ __ORDER_%s_ENDIAN__\n",
1036                 arch_big_endian ? "BIG" : "LITTLE");

1038         add_pre_buffer("#weak_define __PRAGMA_REDEFINE_EXTNAME 1\n");

1040         /*
1041          * This is far from perfect...
1042          */
1043 #ifdef  __sun
1044         add_pre_buffer("#weak_define __unix__ 1\n");
1045         add_pre_buffer("#weak_define __unix 1\n");
1046         add_pre_buffer("#weak_define unix 1\n");
1047         add_pre_buffer("#weak_define __sun__ 1\n");
1048         add_pre_buffer("#weak_define __sun 1\n");
1049         add_pre_buffer("#weak_define sun 1\n");
1050         add_pre_buffer("#weak_define __svr4__ 1\n");
```

```
1051 #endif
1052 }

1054 void declare_builtin_functions(void)
1055 {
1056         /* Gaah. gcc knows tons of builtin <string.h> functions */
1057         add_pre_buffer("extern void *__builtin_memchr(const void *, int, __SIZE_
1058         add_pre_buffer("extern void *__builtin_memcpy(void *, const void *, __SI
1059         add_pre_buffer("extern void *__builtin_mempcpy(void *, const void *, __S
1060         add_pre_buffer("extern void *__builtin_memmove(void *, const void *, __S
1061         add_pre_buffer("extern void *__builtin_memset(void *, int, __SIZE_TYPE__
1062         add_pre_buffer("extern int __builtin_memcmp(const void *, const void *,
1063         add_pre_buffer("extern char *__builtin_strcat(char *, const char *);\n")
1064         add_pre_buffer("extern char *__builtin_strncat(char *, const char *, __S
1065         add_pre_buffer("extern int __builtin_strcmp(const char *, const char *);
1066         add_pre_buffer("extern int __builtin_strncmp(const char *, const char *,
1067         add_pre_buffer("extern int __builtin_strcasecmp(const char *, const char
1068         add_pre_buffer("extern int __builtin_strncasecmp(const char *, const cha
1069         add_pre_buffer("extern char *__builtin_strchr(const char *, int);\n");
1070         add_pre_buffer("extern char *__builtin_strrchr(const char *, int);\n");
1071         add_pre_buffer("extern char *__builtin_strcpy(char *, const char *);\n")
1072         add_pre_buffer("extern char *__builtin_strncpy(char *, const char *, __S
1073         add_pre_buffer("extern char *__builtin_strdup(const char *);\n");
1074         add_pre_buffer("extern char *__builtin_strndup(const char *, __SIZE_TYPE
1075         add_pre_buffer("extern __SIZE_TYPE__ __builtin_strspn(const char *, cons
1076         add_pre_buffer("extern __SIZE_TYPE__ __builtin_strcspn(const char *, con
1077         add_pre_buffer("extern char * __builtin_strpbrk(const char *, const char
1078         add_pre_buffer("extern char* __builtin_stpcpy(const char *, const char*)
1079         add_pre_buffer("extern char* __builtin_stpncpy(const char *, const char*
1080         add_pre_buffer("extern __SIZE_TYPE__ __builtin_strlen(const char *);\n")
1081         add_pre_buffer("extern char *__builtin_strstr(const char *, const char *
1082         add_pre_buffer("extern char *__builtin_strcasestr(const char *, const ch
1083         add_pre_buffer("extern char *__builtin_strnstr(const char *, const char

1085         /* And even some from <strings.h> */
1086         add_pre_buffer("extern int  __builtin_bcmp(const void *, const void *, _
1087         add_pre_buffer("extern void __builtin_bcopy(const void *, void *, __SIZE
1088         add_pre_buffer("extern void __builtin_bzero(void *, __SIZE_TYPE__);\n");
1089         add_pre_buffer("extern char*__builtin_index(const char *, int);\n");
1090         add_pre_buffer("extern char*__builtin_rindex(const char *, int);\n");

1092         /* And bitwise operations.. */
1093         add_pre_buffer("extern int __builtin_clrsb(int);\n");
1094         add_pre_buffer("extern int __builtin_clrsbl(long);\n");
1095         add_pre_buffer("extern int __builtin_clrsbll(long long);\n");
1096         add_pre_buffer("extern int __builtin_clz(int);\n");
1097         add_pre_buffer("extern int __builtin_clzl(long);\n");
1098         add_pre_buffer("extern int __builtin_clzll(long long);\n");
1099         add_pre_buffer("extern int __builtin_ctz(int);\n");
1100         add_pre_buffer("extern int __builtin_ctzl(long);\n");
1101         add_pre_buffer("extern int __builtin_ctzll(long long);\n");
1102         add_pre_buffer("extern int __builtin_ffs(int);\n");
1103         add_pre_buffer("extern int __builtin_ffsl(long);\n");
1104         add_pre_buffer("extern int __builtin_ffsll(long long);\n");
1105         add_pre_buffer("extern int __builtin_parity(unsigned int);\n");
1106         add_pre_buffer("extern int __builtin_parityl(unsigned long);\n");
1107         add_pre_buffer("extern int __builtin_parityll(unsigned long long);\n");
1108         add_pre_buffer("extern int __builtin_popcount(unsigned int);\n");
1109         add_pre_buffer("extern int __builtin_popcountl(unsigned long);\n");
1110         add_pre_buffer("extern int __builtin_popcountll(unsigned long long);\n")

1112         /* And byte swaps.. */
1113         add_pre_buffer("extern unsigned short __builtin_bswap16(unsigned short);
1114         add_pre_buffer("extern unsigned int __builtin_bswap32(unsigned int);\n")
1115         add_pre_buffer("extern unsigned long long __builtin_bswap64(unsigned lon
```



```
1117         /* And atomic memory access functions.. */
1118         add_pre_buffer("extern int __sync_fetch_and_add(void *, ...);\n");
1119         add_pre_buffer("extern int __sync_fetch_and_sub(void *, ...);\n");
1120         add_pre_buffer("extern int __sync_fetch_and_or(void *, ...);\n");
1121         add_pre_buffer("extern int __sync_fetch_and_and(void *, ...);\n");
1122         add_pre_buffer("extern int __sync_fetch_and_xor(void *, ...);\n");
1123         add_pre_buffer("extern int __sync_fetch_and_nand(void *, ...);\n");
1124         add_pre_buffer("extern int __sync_add_and_fetch(void *, ...);\n");
1125         add_pre_buffer("extern int __sync_sub_and_fetch(void *, ...);\n");
1126         add_pre_buffer("extern int __sync_or_and_fetch(void *, ...);\n");
1127         add_pre_buffer("extern int __sync_and_and_fetch(void *, ...);\n");
1128         add_pre_buffer("extern int __sync_xor_and_fetch(void *, ...);\n");
1129         add_pre_buffer("extern int __sync_nand_and_fetch(void *, ...);\n");
1130         add_pre_buffer("extern int __sync_bool_compare_and_swap(void *, ...);\n"
1131         add_pre_buffer("extern int __sync_val_compare_and_swap(void *, ...);\n")
1132         add_pre_buffer("extern void __sync_synchronize();\n");
1133         add_pre_buffer("extern int __sync_lock_test_and_set(void *, ...);\n");
1134         add_pre_buffer("extern void __sync_lock_release(void *, ...);\n");

1136         /* And some random ones.. */
1137         add_pre_buffer("extern void *__builtin_return_address(unsigned int);\n")
1138         add_pre_buffer("extern void *__builtin_extract_return_addr(void *);\n");
1139         add_pre_buffer("extern void *__builtin_frame_address(unsigned int);\n");
1140         add_pre_buffer("extern void __builtin_trap(void);\n");
1141         add_pre_buffer("extern void *__builtin_alloca(__SIZE_TYPE__);\n");
1142         add_pre_buffer("extern void __builtin_prefetch (const void *, ...);\n");
1143         add_pre_buffer("extern long __builtin_alpha_extbl(long, long);\n");
1144         add_pre_buffer("extern long __builtin_alpha_extwl(long, long);\n");
1145         add_pre_buffer("extern long __builtin_alpha_insbl(long, long);\n");
1146         add_pre_buffer("extern long __builtin_alpha_inswl(long, long);\n");
1147         add_pre_buffer("extern long __builtin_alpha_insql(long, long);\n");
1148         add_pre_buffer("extern long __builtin_alpha_inslh(long, long);\n");
1149         add_pre_buffer("extern long __builtin_alpha_cmpbge(long, long);\n");
1150         add_pre_buffer("extern int  __builtin_abs(int);\n");
1151         add_pre_buffer("extern long __builtin_labs(long);\n");
1152         add_pre_buffer("extern long long __builtin_llabs(long long);\n");
1153         add_pre_buffer("extern double __builtin_fabs(double);\n");
1154         add_pre_buffer("extern __SIZE_TYPE__ __builtin_va_arg_pack_len(void);\n"

1156         /* Add Blackfin-specific stuff */
1157         add_pre_buffer(
1158                 "#ifdef __bfin__\n"
1159                 "extern void __builtin_bfin_csync(void);\n"
1160                 "extern void __builtin_bfin_ssync(void);\n"
1161                 "extern int __builtin_bfin_norm_fr1x32(int);\n"
1162                 "#endif\n"
1163         );

1165         /* And some floating point stuff.. */
1166         add_pre_buffer("extern int __builtin_isgreater(float, float);\n");
1167         add_pre_buffer("extern int __builtin_isgreaterequal(float, float);\n");
1168         add_pre_buffer("extern int __builtin_isless(float, float);\n");
1169         add_pre_buffer("extern int __builtin_islessequal(float, float);\n");
1170         add_pre_buffer("extern int __builtin_islessgreater(float, float);\n");
1171         add_pre_buffer("extern int __builtin_isunordered(float, float);\n");

1173         /* And some INFINITY / NAN stuff.. */
1174         add_pre_buffer("extern double __builtin_huge_val(void);\n");
1175         add_pre_buffer("extern float __builtin_huge_valf(void);\n");
1176         add_pre_buffer("extern long double __builtin_huge_vall(void);\n");
1177         add_pre_buffer("extern double __builtin_inf(void);\n");
1178         add_pre_buffer("extern float __builtin_inff(void);\n");
1179         add_pre_buffer("extern long double __builtin_infl(void);\n");
1180         add_pre_buffer("extern double __builtin_nan(const char *);\n");
1181         add_pre_buffer("extern float __builtin_nanf(const char *);\n");
1182         add_pre_buffer("extern long double __builtin_nanl(const char *);\n");
```

```
1183         add_pre_buffer("extern int __builtin_isinf_sign(float);\n");
1184         add_pre_buffer("extern int __builtin_isfinite(float);\n");
1185         add_pre_buffer("extern int __builtin_isnan(float);\n");

1187         /* And some __FORTIFY_SOURCE ones.. */
1188         add_pre_buffer ("extern __SIZE_TYPE__ __builtin_object_size(const void *
1189         add_pre_buffer ("extern void * __builtin___memcpy_chk(void *, const void
1190         add_pre_buffer ("extern void * __builtin___memmove_chk(void *, const voi
1191         add_pre_buffer ("extern void * __builtin___mempcpy_chk(void *, const voi
1192         add_pre_buffer ("extern void * __builtin___memset_chk(void *, int, __SIZ
1193         add_pre_buffer ("extern int __builtin___sprintf_chk(char *, int, __SIZE_
1194         add_pre_buffer ("extern int __builtin___snprintf_chk(char *, __SIZE_TYPE
1195         add_pre_buffer ("extern char * __builtin___stpcpy_chk(char *, const char
1196         add_pre_buffer ("extern char * __builtin___strcat_chk(char *, const char
1197         add_pre_buffer ("extern char * __builtin___strcpy_chk(char *, const char
1198         add_pre_buffer ("extern char * __builtin___strncat_chk(char *, const cha
1199         add_pre_buffer ("extern char * __builtin___strncpy_chk(char *, const cha
1200         add_pre_buffer ("extern int __builtin___vsprintf_chk(char *, int, __SIZE
1201         add_pre_buffer ("extern int __builtin___vsnprintf_chk(char *, __SIZE_TYP
1202         add_pre_buffer ("extern void __builtin_unreachable(void);\n");

1204         /* And some from <stdlib.h> */
1205         add_pre_buffer("extern void __builtin_abort(void);\n");
1206         add_pre_buffer("extern void *__builtin_calloc(__SIZE_TYPE__, __SIZE_TYPE
1207         add_pre_buffer("extern void __builtin_exit(int);\n");
1208         add_pre_buffer("extern void *__builtin_malloc(__SIZE_TYPE__);\n");
1209         add_pre_buffer("extern void *__builtin_realloc(void *, __SIZE_TYPE__);\n
1210         add_pre_buffer("extern void __builtin_free(void *);\n");

1212         /* And some from <stdio.h> */
1213         add_pre_buffer("extern int __builtin_printf(const char *, ...);\n");
1214         add_pre_buffer("extern int __builtin_sprintf(char *, const char *, ...);
1215         add_pre_buffer("extern int __builtin_snprintf(char *, __SIZE_TYPE__, con
1216         add_pre_buffer("extern int __builtin_puts(const char *);\n");
1217         add_pre_buffer("extern int __builtin_vprintf(const char *, __builtin_va_
1218         add_pre_buffer("extern int __builtin_vsprintf(char *, const char *, __bu
1219         add_pre_buffer("extern int __builtin_vsnprintf(char *, __SIZE_TYPE__, co
1220 }

1222 void create_builtin_stream(void)
1223 {
1224         add_pre_buffer("#weak_define __GNUC__ %d\n", gcc_major);
1225         add_pre_buffer("#weak_define __GNUC_MINOR__ %d\n", gcc_minor);
1226         add_pre_buffer("#weak_define __GNUC_PATCHLEVEL__ %d\n", gcc_patchlevel);

1228         /* add the multiarch include directories, if any */
1229         if (multiarch_dir && *multiarch_dir) {
1230                 add_pre_buffer("#add_system \"/usr/include/%s\"\n", multiarch_di
1231                 add_pre_buffer("#add_system \"/usr/local/include/%s\"\n", multia
1232         }

1234         /* We add compiler headers path here because we have to parse
1235          * the arguments to get it, falling back to default. */
1236         add_pre_buffer("#add_system \"%s/include\"\n", gcc_base_dir);
1237         add_pre_buffer("#add_system \"%s/include-fixed\"\n", gcc_base_dir);

1239         add_pre_buffer("#define __extension__\n");
1240         add_pre_buffer("#define __pragma__\n");
1241         add_pre_buffer("#define _Pragma(x)\n");

1243         // gcc defines __SIZE_TYPE__ to be size_t.  For linux/i86 and
1244         // solaris/sparc that is really "unsigned int" and for linux/x86_64
1245         // it is "long unsigned int".  In either case we can probably
1246         // get away with this.  We need the #weak_define as cgcc will define
1247         // the right __SIZE_TYPE__.
1248         if (size_t_ctype == &ulong_ctype)
```



```
1249                 add_pre_buffer("#weak_define __SIZE_TYPE__ long unsigned int\n")
1250         else
1251                 add_pre_buffer("#weak_define __SIZE_TYPE__ unsigned int\n");
1252         add_pre_buffer("#weak_define __STDC__ 1\n");

1254         switch (standard)
1255         {
1256                 case STANDARD_C89:
1257                         add_pre_buffer("#weak_define __STRICT_ANSI__\n");
1258                         break;

1260                 case STANDARD_C94:
1261                         add_pre_buffer("#weak_define __STDC_VERSION__ 199409L\n"
1262                         add_pre_buffer("#weak_define __STRICT_ANSI__\n");
1263                         break;

1265                 case STANDARD_C99:
1266                         add_pre_buffer("#weak_define __STDC_VERSION__ 199901L\n"
1267                         add_pre_buffer("#weak_define __STRICT_ANSI__\n");
1268                         break;

1270                 case STANDARD_GNU89:
1271                         break;

1273                 case STANDARD_GNU99:
1274                         add_pre_buffer("#weak_define __STDC_VERSION__ 199901L\n"
1275                         break;

1277                 case STANDARD_C11:
1278                         add_pre_buffer("#weak_define __STRICT_ANSI__ 1\n");
1279                 case STANDARD_GNU11:
1280                         add_pre_buffer("#weak_define __STDC_NO_ATOMICS__ 1\n");
1281                         add_pre_buffer("#weak_define __STDC_NO_COMPLEX__ 1\n");
1282                         add_pre_buffer("#weak_define __STDC_NO_THREADS__ 1\n");
1283                         add_pre_buffer("#weak_define __STDC_VERSION__ 201112L\n"
1284                         break;

1286                 default:
1287                         assert (0);
1288         }

1290         add_pre_buffer("#define __builtin_stdarg_start(a,b) ((a) = (__builtin_va
1291         add_pre_buffer("#define __builtin_va_start(a,b) ((a) = (__builtin_va_lis
1292         add_pre_buffer("#define __builtin_ms_va_start(a,b) ((a) = (__builtin_ms_
1293         add_pre_buffer("#define __builtin_va_arg(arg,type)  ({ type __va_arg_ret
1294         add_pre_buffer("#define __builtin_va_alist (*(void *)0)\n");
1295         add_pre_buffer("#define __builtin_va_arg_incr(x) ((x) + 1)\n");
1296         add_pre_buffer("#define __builtin_va_copy(dest, src) ({ dest = src; (voi
1297         add_pre_buffer("#define __builtin_ms_va_copy(dest, src) ({ dest = src; (
1298         add_pre_buffer("#define __builtin_va_end(arg)\n");
1299         add_pre_buffer("#define __builtin_ms_va_end(arg)\n");
1300         add_pre_buffer("#define __builtin_va_arg_pack()\n");

1302         /* FIXME! We need to do these as special magic macros at expansion time!
1303         add_pre_buffer("#define __BASE_FILE__ \"base_file.c\"\n");

1305         if (optimize)
1306                 add_pre_buffer("#define __OPTIMIZE__ 1\n");
1307         if (optimize_size)
1308                 add_pre_buffer("#define __OPTIMIZE_SIZE__ 1\n");
1309 }

1311 static struct symbol_list *sparse_tokenstream(struct token *token)
1312 {
1313         int builtin = token && !token->pos.stream;
```

```
1315         // Preprocess the stream
1316         token = preprocess(token);
1317
1318         if (dump_macro_defs && !builtin)
1319                 dump_macro_definitions();
1320
1321         if (preprocess_only) {
1322                 while (!eof_token(token)) {
1323                         int prec = 1;
1324                         struct token *next = token->next;
1325                         const char *separator = "";
1326                         if (next->pos.whitespace)
1327                                 separator = " ";
1328                         if (next->pos.newline) {
1329                                 separator = "\n\t\t\t\t\t";
1330                                 prec = next->pos.pos;
1331                                 if (prec > 4)
1332                                         prec = 4;
1333                         }
1334                         printf("%s%.*s", show_token(token), prec, separator);
1335                         token = next;
1336                 }
1337                 putchar('\n');
1338
1339                 return NULL;
1340         }
1341
1342         // Parse the resulting C code
1343         while (!eof_token(token))
1344                 token = external_declaration(token, &translation_unit_used_list,
1345         return translation_unit_used_list;
1346 }
1347
1348 static struct symbol_list *sparse_file(const char *filename)
1349 {
1350         int fd;
1351         struct token *token;
1352
1353         if (strcmp (filename, "-") == 0) {
1354                 fd = 0;
1355         } else {
1356                 fd = open(filename, O_RDONLY);
1357                 if (fd < 0)
1358                         die("No such file: %s", filename);
1359         }
1360
1361         // Tokenize the input stream
1362         token = tokenize(filename, fd, NULL, includepath);
1363         store_all_tokens(token);
1364         close(fd);
1365
1366         return sparse_tokenstream(token);
1367 }
1368
1369 static int endswith(const char *str, const char *suffix)
1370 {
1371         const char *found = strstr(str, suffix);
1372         return (found && strcmp(found, suffix) == 0);
1373 }
1374
1375 /*
1376  * This handles the "-include" directive etc: we're in global
1377  * scope, and all types/macros etc will affect all the following
1378  * files.
1379  *
1380  * NOTE NOTE NOTE! "#undef" of anything in this stage will
```



```
1381  * affect all subsequent files too, i.e. we can have non-local
1382  * behaviour between files!
1383  */
1384 static struct symbol_list *sparse_initial(void)
1385 {
1386         int i;
1387
1388         // Prepend any "include" file to the stream.
1389         // We're in global scope, it will affect all files!
1390         for (i = 0; i < cmdline_include_nr; i++)
1391                 add_pre_buffer("#argv_include \"%s\"\n", cmdline_include[i]);
1392
1393         return sparse_tokenstream(pre_buffer_begin);
1394 }
1395
1396 struct symbol_list *sparse_initialize(int argc, char **argv, struct string_list
1397 {
1398         char **args;
1399         struct symbol_list *list;
1400
1401         // Initialize symbol stream first, so that we can add defines etc
1402         init_symbols();
1403         init_include_path();
1404
1405         progname = argv[0];
1406
1407         args = argv;
1408         for (;;) {
1409                 char *arg = *++args;
1410                 if (!arg)
1411                         break;
1412
1413                 if (arg[0] == '-' && arg[1]) {
1414                         args = handle_switch(arg+1, args);
1415                         continue;
1416                 }
1417
1418                 if (endswith(arg, ".a") || endswith(arg, ".so") ||
1419                     endswith(arg, ".so.1") || endswith(arg, ".o"))
1420                         continue;
1421
1422                 add_ptr_list_notag(filelist, arg);
1423         }
1424         handle_switch_W_finalize();
1425         handle_switch_v_finalize();
1426
1427         handle_arch_finalize();
1428
1429         list = NULL;
1430         if (!ptr_list_empty(filelist)) {
1431                 // Initialize type system
1432                 init_ctype();
1433                 handle_funsigned_char();
1434
1435                 create_builtin_stream();
1436                 predefined_macros();
1437                 if (!preprocess_only)
1438                         declare_builtin_functions();
1439
1440                 list = sparse_initial();
1441
1442                 /*
1443                  * Protect the initial token allocations, since
1444                  * they need to survive all the others
1445                  */
1446                 protect_token_alloc();
```

```
1447         }
1448         /*
1449          * Evaluate the complete symbol list
1450          * Note: This is not needed for normal cases.
1451          *       These symbols should only be predefined defines and
1452          *       declaratons which will be evaluated later, when needed.
1453          *       This is also the case when a file is directly included via
1454          *       '-include <file>' on the command line *AND* the file only
1455          *       contains defines, declarations and inline definitions.
1456          *       However, in the rare cases where the given file should
1457          *       contain some definitions, these will never be evaluated
1458          *       and thus won't be able to be linearized correctly.
1459          *       Hence the evaluate_symbol_list() here under.
1460          */
1461         evaluate_symbol_list(list);
1462         return list;
1463 }

1465 struct symbol_list * sparse_keep_tokens(char *filename)
1466 {
1467         struct symbol_list *res;

1469         /* Clear previous symbol list */
1470         translation_unit_used_list = NULL;

1472         new_file_scope();
1473         res = sparse_file(filename);

1475         /* And return it */
1476         return res;
1477 }


1480 struct symbol_list * __sparse(char *filename)
1481 {
1482         struct symbol_list *res;

1484         res = sparse_keep_tokens(filename);

1486         /* Drop the tokens for this file after parsing */
1487         clear_token_alloc();

1489         /* And return it */
1490         return res;
1491 }

1493 struct symbol_list * sparse(char *filename)
1494 {
1495         struct symbol_list *res = __sparse(filename);

1497         if (has_error & ERROR_CURR_PHASE)
1498                 has_error = ERROR_PREV_PHASE;
1499         /* Evaluate the complete symbol list */
1500         evaluate_symbol_list(res);

1502         return res;
1503 }
```

```
**********************************************************
    8205 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/lib.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef LIB_H
#define LIB_H

#include <stdlib.h>
#include <stddef.h>

/*
 * Basic helper routine descriptions for 'sparse'.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *               2004 Christopher Li
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include "compat.h"
#include "ptrlist.h"

#define DO_STRINGIFY(x) #x
#define STRINGIFY(x) DO_STRINGIFY(x)

#ifndef ARRAY_SIZE
#define ARRAY_SIZE(x) (sizeof(x)/sizeof((x)[0]))
#endif

extern int verbose, optimize, optimize_size, preprocessing;
extern int die_if_error;
extern int parse_error;
extern int repeat_phase, merge_phi_sources;
extern int gcc_major, gcc_minor, gcc_patchlevel;

extern unsigned int hexval(unsigned int c);

struct position {
        unsigned int type:6,
                     stream:14,
                     newline:1,
                     whitespace:1,
                     pos:10;
        unsigned int line:31,
                     noexpand:1;
};
```



```
struct ident;
struct token;
struct symbol;
struct statement;
struct expression;
struct basic_block;
struct entrypoint;
struct instruction;
struct multijmp;
struct pseudo;

DECLARE_PTR_LIST(symbol_list, struct symbol);
DECLARE_PTR_LIST(statement_list, struct statement);
DECLARE_PTR_LIST(expression_list, struct expression);
DECLARE_PTR_LIST(basic_block_list, struct basic_block);
DECLARE_PTR_LIST(instruction_list, struct instruction);
DECLARE_PTR_LIST(multijmp_list, struct multijmp);
DECLARE_PTR_LIST(pseudo_list, struct pseudo);
DECLARE_PTR_LIST(ident_list, struct ident);
DECLARE_PTR_LIST(string_list, char);

typedef struct pseudo *pseudo_t;

struct token *skip_to(struct token *, int);
struct token *expect(struct token *, int, const char *);
#ifdef __GNUC__
#define FORMAT_ATTR(pos) __attribute__ ((__format__ (__printf__, pos, pos+1)))
#define NORETURN_ATTR __attribute__ ((__noreturn__))
#define SENTINEL_ATTR __attribute__ ((__sentinel__))
#else
#define FORMAT_ATTR(pos)
#define NORETURN_ATTR
#define SENTINEL_ATTR
#endif

FORMAT_ATTR(1) NORETURN_ATTR
extern void die(const char *, ...);

FORMAT_ATTR(2) NORETURN_ATTR
extern void error_die(struct position, const char *, ...);

extern void info(struct position, const char *, ...) FORMAT_ATTR(2);
extern void warning(struct position, const char *, ...) FORMAT_ATTR(2);
extern void sparse_error(struct position, const char *, ...) FORMAT_ATTR(2);
extern void expression_error(struct expression *, const char *, ...) FORMAT_ATTR

#define ERROR_CURR_PHASE        (1 << 0)
#define ERROR_PREV_PHASE        (1 << 1)
extern int has_error;

extern void add_pre_buffer(const char *fmt, ...) FORMAT_ATTR(1);

extern int preprocess_only;

extern int Waddress;
extern int Waddress_space;
extern int Wbitwise;
extern int Wcast_to_as;
extern int Wcast_truncate;
extern int Wconstant_suffix;
extern int Wconstexpr_not_const;
extern int Wcontext;
extern int Wdecl;
extern int Wdeclarationafterstatement;
extern int Wdefault_bitfield_sign;
extern int Wdesignated_init;
```

```
127 extern int Wdo_while;
128 extern int Wenum_mismatch;
129 extern int Wexternal_function_has_definition;
130 extern int Wsparse_error;
131 extern int Wimplicit_int;
132 extern int Winit_cstring;
133 extern int Wmemcpy_max_count;
134 extern int Wnon_pointer_null;
135 extern int Wold_initializer;
136 extern int Wold_style_definition;
137 extern int Wone_bit_signed_bitfield;
138 extern int Woverride_init;
139 extern int Woverride_init_all;
140 extern int Woverride_init_whole_range;
141 extern int Wparen_string;
142 extern int Wpointer_arith;
143 extern int Wptr_subtraction_blows;
144 extern int Wreturn_void;
145 extern int Wshadow;
146 extern int Wsizeof_bool;
147 extern int Wstrict_prototypes;
148 extern int Wtautological_compare;
149 extern int Wtransparent_union;
150 extern int Wtypesign;
151 extern int Wundef;
152 extern int Wuninitialized;
153 extern int Wunknown_attribute;
154 extern int Wvla;

156 extern int dump_macro_defs;

158 extern int dbg_entry;
159 extern int dbg_dead;

161 extern int fmem_report;
162 extern int fdump_linearize;
163 extern unsigned long long fmemcpy_max_count;

165 extern int arch_m64;
166 extern int arch_msize_long;
167 extern int arch_big_endian;

169 extern void declare_builtin_functions(void);
170 extern void create_builtin_stream(void);
171 extern void dump_macro_definitions(void);
172 extern struct symbol_list *sparse_initialize(int argc, char **argv, struct strin
173 extern struct symbol_list *__sparse(char *filename);
174 extern struct symbol_list *sparse_keep_tokens(char *filename);
175 extern struct symbol_list *sparse(char *filename);
176 extern void report_stats(void);

178 static inline int symbol_list_size(struct symbol_list *list)
179 {
180         return ptr_list_size((struct ptr_list *)(list));
181 }

183 static inline int statement_list_size(struct statement_list *list)
184 {
185         return ptr_list_size((struct ptr_list *)(list));
186 }

188 static inline int expression_list_size(struct expression_list *list)
189 {
190         return ptr_list_size((struct ptr_list *)(list));
191 }
```



```
193 static inline int instruction_list_size(struct instruction_list *list)
194 {
195         return ptr_list_size((struct ptr_list *)(list));
196 }

198 static inline int pseudo_list_size(struct pseudo_list *list)
199 {
200         return ptr_list_size((struct ptr_list *)(list));
201 }

203 static inline int bb_list_size(struct basic_block_list *list)
204 {
205         return ptr_list_size((struct ptr_list *)(list));
206 }

208 static inline void free_instruction_list(struct instruction_list **head)
209 {
210         free_ptr_list((struct ptr_list **)head);
211 }

213 static inline struct instruction * delete_last_instruction(struct instruction_li
214 {
215         return undo_ptr_list_last((struct ptr_list **)head);
216 }

218 static inline struct basic_block * delete_last_basic_block(struct basic_block_li
219 {
220         return delete_ptr_list_last((struct ptr_list **)head);
221 }

223 static inline struct basic_block *first_basic_block(struct basic_block_list *hea
224 {
225         return first_ptr_list((struct ptr_list *)head);
226 }
227 static inline struct instruction *last_instruction(struct instruction_list *head
228 {
229         return last_ptr_list((struct ptr_list *)head);
230 }

232 static inline struct instruction *first_instruction(struct instruction_list *hea
233 {
234         return first_ptr_list((struct ptr_list *)head);
235 }

237 static inline struct expression *first_expression(struct expression_list *head)
238 {
239         return first_ptr_list((struct ptr_list *)head);
240 }

242 static inline pseudo_t first_pseudo(struct pseudo_list *head)
243 {
244         return first_ptr_list((struct ptr_list *)head);
245 }

247 static inline void concat_symbol_list(struct symbol_list *from, struct symbol_li
248 {
249         concat_ptr_list((struct ptr_list *)from, (struct ptr_list **)to);
250 }

252 static inline void concat_basic_block_list(struct basic_block_list *from, struct
253 {
254         concat_ptr_list((struct ptr_list *)from, (struct ptr_list **)to);
255 }

257 static inline void concat_instruction_list(struct instruction_list *from, struct
258 {
```

```
259         concat_ptr_list((struct ptr_list *)from, (struct ptr_list **)to);
260 }

262 static inline void add_symbol(struct symbol_list **list, struct symbol *sym)
263 {
264         add_ptr_list(list, sym);
265 }

267 static inline void add_statement(struct statement_list **list, struct statement
268 {
269         add_ptr_list(list, stmt);
270 }

272 static inline void add_expression(struct expression_list **list, struct expressi
273 {
274         add_ptr_list(list, expr);
275 }

277 static inline void add_ident(struct ident_list **list, struct ident *ident)
278 {
279         add_ptr_list(list, ident);
280 }

282 #define hashval(x) ((unsigned long)(x))

284 #endif
```

```
**********************************************************
   59041 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/linearize.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Linearize - walk the statement tree (but _not_ the expressions)
 * to generate a linear version of it and the basic blocks.
 *
 * NOTE! We're not interested in the actual sub-expressions yet,
 * even though they can generate conditional branches and
 * subroutine calls. That's all "local" behaviour.
 *
 * Copyright (C) 2004 Linus Torvalds
 * Copyright (C) 2004 Christopher Li
 */

#include <string.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>

#include "parse.h"
#include "expression.h"
#include "linearize.h"
#include "flow.h"
#include "target.h"

pseudo_t linearize_statement(struct entrypoint *ep, struct statement *stmt);
pseudo_t linearize_expression(struct entrypoint *ep, struct expression *expr);

static pseudo_t add_binary_op(struct entrypoint *ep, struct symbol *ctype, int o
static pseudo_t add_setval(struct entrypoint *ep, struct symbol *ctype, struct e
static pseudo_t linearize_one_symbol(struct entrypoint *ep, struct symbol *sym);

struct access_data;
static pseudo_t add_load(struct entrypoint *ep, struct access_data *);
static pseudo_t linearize_initializer(struct entrypoint *ep, struct expression *
static pseudo_t cast_pseudo(struct entrypoint *ep, pseudo_t src, struct symbol *

struct pseudo void_pseudo = {};

static struct position current_pos;

ALLOCATOR(pseudo_user, "pseudo_user");

static struct instruction *alloc_instruction(int opcode, int size)
{
        struct instruction * insn = __alloc_instruction(0);
        insn->opcode = opcode;
        insn->size = size;
        insn->pos = current_pos;
        return insn;
}

static inline int type_size(struct symbol *type)
{
        return type ? type->bit_size > 0 ? type->bit_size : 0 : 0;
}

static struct instruction *alloc_typed_instruction(int opcode, struct symbol *ty
{
        struct instruction *insn = alloc_instruction(opcode, type_size(type));
        insn->type = type;
```



```
        return insn;
}

static struct entrypoint *alloc_entrypoint(void)
{
        return __alloc_entrypoint(0);
}

static struct basic_block *alloc_basic_block(struct entrypoint *ep, struct posit
{
        static int nr;
        struct basic_block *bb = __alloc_basic_block(0);
        bb->context = -1;
        bb->pos = pos;
        bb->ep = ep;
        bb->nr = nr++;
        return bb;
}

static struct multijmp *alloc_multijmp(struct basic_block *target, int begin, in
{
        struct multijmp *multijmp = __alloc_multijmp(0);
        multijmp->target = target;
        multijmp->begin = begin;
        multijmp->end = end;
        return multijmp;
}

static inline int regno(pseudo_t n)
{
        int retval = -1;
        if (n && n->type == PSEUDO_REG)
                retval = n->nr;
        return retval;
}

const char *show_pseudo(pseudo_t pseudo)
{
        static int n;
        static char buffer[4][64];
        char *buf;
        int i;

        if (!pseudo)
                return "no pseudo";
        if (pseudo == VOID)
                return "VOID";
        buf = buffer[3 & ++n];
        switch(pseudo->type) {
        case PSEUDO_SYM: {
                struct symbol *sym = pseudo->sym;
                struct expression *expr;

                if (sym->bb_target) {
                        snprintf(buf, 64, ".L%u", sym->bb_target->nr);
                        break;
                }
                if (sym->ident) {
                        snprintf(buf, 64, "%s", show_ident(sym->ident));
                        break;
                }
                expr = sym->initializer;
                snprintf(buf, 64, "<anon symbol:%p>", sym);
                if (expr) {
                        switch (expr->type) {
                        case EXPR_VALUE:
```

```
127                                 snprintf(buf, 64, "<symbol value: %lld>", expr->
128                                 break;
129                         case EXPR_STRING:
130                                 return show_string(expr->string);
131                         default:
132                                 break;
133                         }
134                 }
135                 break;
136         }
137         case PSEUDO_REG:
138                 i = snprintf(buf, 64, "%%r%d", pseudo->nr);
139                 if (pseudo->ident)
140                         sprintf(buf+i, "(%s)", show_ident(pseudo->ident));
141                 break;
142         case PSEUDO_VAL: {
143                 long long value = pseudo->value;
144                 if (value > 1000 || value < -1000)
145                         snprintf(buf, 64, "$%#llx", value);
146                 else
147                         snprintf(buf, 64, "$%lld", value);
148                 break;
149         }
150         case PSEUDO_ARG:
151                 snprintf(buf, 64, "%%arg%d", pseudo->nr);
152                 break;
153         case PSEUDO_PHI:
154                 i = snprintf(buf, 64, "%%phi%d", pseudo->nr);
155                 if (pseudo->ident)
156                         sprintf(buf+i, "(%s)", show_ident(pseudo->ident));
157                 break;
158         default:
159                 snprintf(buf, 64, "<bad pseudo type %d>", pseudo->type);
160         }
161         return buf;
162 }

164 static const char *opcodes[] = {
165         [OP_BADOP] = "bad_op",

167         /* Fn entrypoint */
168         [OP_ENTRY] = "<entry-point>",

170         /* Terminator */
171         [OP_RET] = "ret",
172         [OP_BR] = "br",
173         [OP_CBR] = "cbr",
174         [OP_SWITCH] = "switch",
175         [OP_INVOKE] = "invoke",
176         [OP_COMPUTEDGOTO] = "jmp *",
177         [OP_UNWIND] = "unwind",
178
179         /* Binary */
180         [OP_ADD] = "add",
181         [OP_SUB] = "sub",
182         [OP_MULU] = "mulu",
183         [OP_MULS] = "muls",
184         [OP_DIVU] = "divu",
185         [OP_DIVS] = "divs",
186         [OP_MODU] = "modu",
187         [OP_MODS] = "mods",
188         [OP_SHL] = "shl",
189         [OP_LSR] = "lsr",
190         [OP_ASR] = "asr",
191
192         /* Logical */
```



```
193         [OP_AND] = "and",
194         [OP_OR] = "or",
195         [OP_XOR] = "xor",
196         [OP_AND_BOOL] = "and-bool",
197         [OP_OR_BOOL] = "or-bool",

199         /* Binary comparison */
200         [OP_SET_EQ] = "seteq",
201         [OP_SET_NE] = "setne",
202         [OP_SET_LE] = "setle",
203         [OP_SET_GE] = "setge",
204         [OP_SET_LT] = "setlt",
205         [OP_SET_GT] = "setgt",
206         [OP_SET_B] = "setb",
207         [OP_SET_A] = "seta",
208         [OP_SET_BE] = "setbe",
209         [OP_SET_AE] = "setae",

211         /* Uni */
212         [OP_NOT] = "not",
213         [OP_NEG] = "neg",

215         /* Special three-input */
216         [OP_SEL] = "select",
217
218         /* Memory */
219         [OP_MALLOC] = "malloc",
220         [OP_FREE] = "free",
221         [OP_ALLOCA] = "alloca",
222         [OP_LOAD] = "load",
223         [OP_STORE] = "store",
224         [OP_SETVAL] = "set",
225         [OP_SYMADDR] = "symaddr",
226         [OP_GET_ELEMENT_PTR] = "getelem",

228         /* Other */
229         [OP_PHI] = "phi",
230         [OP_PHISOURCE] = "phisrc",
231         [OP_CAST] = "cast",
232         [OP_SCAST] = "scast",
233         [OP_FPCAST] = "fpcast",
234         [OP_PTRCAST] = "ptrcast",
235         [OP_INLINED_CALL] = "# call",
236         [OP_CALL] = "call",
237         [OP_VANEXT] = "va_next",
238         [OP_VAARG] = "va_arg",
239         [OP_SLICE] = "slice",
240         [OP_SNOP] = "snop",
241         [OP_LNOP] = "lnop",
242         [OP_NOP] = "nop",
243         [OP_DEATHNOTE] = "dead",
244         [OP_ASM] = "asm",

246         /* Sparse tagging (line numbers, context, whatever) */
247         [OP_CONTEXT] = "context",
248         [OP_RANGE] = "range-check",

250         [OP_COPY] = "copy",
251 };

253 static char *show_asm_constraints(char *buf, const char *sep, struct asm_constra
254 {
255         struct asm_constraint *entry;

257         FOR_EACH_PTR(list, entry) {
258                 buf += sprintf(buf, "%s\"%s\"", sep, entry->constraint);
```

```
259                 if (entry->pseudo)
260                         buf += sprintf(buf, " (%s)", show_pseudo(entry->pseudo))
261                 if (entry->ident)
262                         buf += sprintf(buf, " [%s]", show_ident(entry->ident));
263                 sep = ", ";
264         } END_FOR_EACH_PTR(entry);
265         return buf;
266 }

268 static char *show_asm(char *buf, struct instruction *insn)
269 {
270         struct asm_rules *rules = insn->asm_rules;

272         buf += sprintf(buf, "\"%s\"", insn->string);
273         buf = show_asm_constraints(buf, "\n\t\tout: ", rules->outputs);
274         buf = show_asm_constraints(buf, "\n\t\tin: ", rules->inputs);
275         buf = show_asm_constraints(buf, "\n\t\tclobber: ", rules->clobbers);
276         return buf;
277 }

279 const char *show_instruction(struct instruction *insn)
280 {
281         int opcode = insn->opcode;
282         static char buffer[4096];
283         char *buf;

285         buf = buffer;
286         if (!insn->bb)
287                 buf += sprintf(buf, "# ");

289         if (opcode < ARRAY_SIZE(opcodes)) {
290                 const char *op = opcodes[opcode];
291                 if (!op)
292                         buf += sprintf(buf, "opcode:%d", opcode);
293                 else
294                         buf += sprintf(buf, "%s", op);
295                 if (insn->size)
296                         buf += sprintf(buf, ".%d", insn->size);
297                 memset(buf, ' ', 20);
298                 buf++;
299         }

301         if (buf < buffer + 12)
302                 buf = buffer + 12;
303         switch (opcode) {
304         case OP_RET:
305                 if (insn->src && insn->src != VOID)
306                         buf += sprintf(buf, "%s", show_pseudo(insn->src));
307                 break;

309         case OP_CBR:
310                 buf += sprintf(buf, "%s, .L%u, .L%u", show_pseudo(insn->cond), i
311                 break;

313         case OP_BR:
314                 buf += sprintf(buf, ".L%u", insn->bb_true->nr);
315                 break;

317         case OP_SYMADDR: {
318                 struct symbol *sym = insn->symbol->sym;
319                 buf += sprintf(buf, "%s <- ", show_pseudo(insn->target));

321                 if (!insn->bb && !sym)
322                         break;
323                 if (sym->bb_target) {
324                         buf += sprintf(buf, ".L%u", sym->bb_target->nr);
```



```
325                         break;
326                 }
327                 if (sym->ident) {
328                         buf += sprintf(buf, "%s", show_ident(sym->ident));
329                         break;
330                 }
331                 buf += sprintf(buf, "<anon symbol:%p>", sym);
332                 break;
333         }
334
335         case OP_SETVAL: {
336                 struct expression *expr = insn->val;
337                 struct symbol *sym;
338                 buf += sprintf(buf, "%s <- ", show_pseudo(insn->target));

340                 if (!expr) {
341                         buf += sprintf(buf, "%s", "<none>");
342                         break;
343                 }

345                 switch (expr->type) {
346                 case EXPR_VALUE:
347                         buf += sprintf(buf, "%lld", expr->value);
348                         break;
349                 case EXPR_FVALUE:
350                         buf += sprintf(buf, "%Lf", expr->fvalue);
351                         break;
352                 case EXPR_STRING:
353                         buf += sprintf(buf, "%.40s", show_string(expr->string));
354                         break;
355                 case EXPR_SYMBOL:
356                         buf += sprintf(buf, "%s", show_ident(expr->symbol->ident
357                         break;
358                 case EXPR_LABEL:
359                         sym = expr->symbol;
360                         if (sym->bb_target)
361                                 buf += sprintf(buf, ".L%u", sym->bb_target->nr);
362                         break;
363                 default:
364                         buf += sprintf(buf, "SETVAL EXPR TYPE %d", expr->type);
365                 }
366                 break;
367         }
368         case OP_SWITCH: {
369                 struct multijmp *jmp;
370                 buf += sprintf(buf, "%s", show_pseudo(insn->cond));
371                 FOR_EACH_PTR(insn->multijmp_list, jmp) {
372                         if (jmp->begin == jmp->end)
373                                 buf += sprintf(buf, ", %d -> .L%u", jmp->begin,
374                         else if (jmp->begin < jmp->end)
375                                 buf += sprintf(buf, ", %d ... %d -> .L%u", jmp->
376                         else
377                                 buf += sprintf(buf, ", default -> .L%u", jmp->ta
378                 } END_FOR_EACH_PTR(jmp);
379                 break;
380         }
381         case OP_COMPUTEDGOTO: {
382                 struct multijmp *jmp;
383                 buf += sprintf(buf, "%s", show_pseudo(insn->target));
384                 FOR_EACH_PTR(insn->multijmp_list, jmp) {
385                         buf += sprintf(buf, ", .L%u", jmp->target->nr);
386                 } END_FOR_EACH_PTR(jmp);
387                 break;
388         }

390         case OP_PHISOURCE: {
```

```
391                 struct instruction *phi;
392                 buf += sprintf(buf, "%s <- %s    ", show_pseudo(insn->target), s
393                 FOR_EACH_PTR(insn->phi_users, phi) {
394                         buf += sprintf(buf, " (%s)", show_pseudo(phi->target));
395                 } END_FOR_EACH_PTR(phi);
396                 break;
397         }

399         case OP_PHI: {
400                 pseudo_t phi;
401                 const char *s = " <-";
402                 buf += sprintf(buf, "%s", show_pseudo(insn->target));
403                 FOR_EACH_PTR(insn->phi_list, phi) {
404                         buf += sprintf(buf, "%s %s", s, show_pseudo(phi));
405                         s = ",";
406                 } END_FOR_EACH_PTR(phi);
407                 break;
408         }
409         case OP_LOAD: case OP_LNOP:
410                 buf += sprintf(buf, "%s <- %d[%s]", show_pseudo(insn->target), i
411                 break;
412         case OP_STORE: case OP_SNOP:
413                 buf += sprintf(buf, "%s -> %d[%s]", show_pseudo(insn->target), i
414                 break;
415         case OP_INLINED_CALL:
416         case OP_CALL: {
417                 struct pseudo *arg;
418                 if (insn->target && insn->target != VOID)
419                         buf += sprintf(buf, "%s <- ", show_pseudo(insn->target))
420                 buf += sprintf(buf, "%s", show_pseudo(insn->func));
421                 FOR_EACH_PTR(insn->arguments, arg) {
422                         buf += sprintf(buf, ", %s", show_pseudo(arg));
423                 } END_FOR_EACH_PTR(arg);
424                 break;
425         }
426         case OP_CAST:
427         case OP_SCAST:
428         case OP_FPCAST:
429         case OP_PTRCAST:
430                 buf += sprintf(buf, "%s <- (%d) %s",
431                         show_pseudo(insn->target),
432                         type_size(insn->orig_type),
433                         show_pseudo(insn->src));
434                 break;
435         case OP_BINARY ... OP_BINARY_END:
436         case OP_BINCMP ... OP_BINCMP_END:
437                 buf += sprintf(buf, "%s <- %s, %s", show_pseudo(insn->target), s
438                 break;

440         case OP_SEL:
441                 buf += sprintf(buf, "%s <- %s, %s, %s", show_pseudo(insn->target
442                         show_pseudo(insn->src1), show_pseudo(insn->src2), show_p
443                 break;

445         case OP_SLICE:
446                 buf += sprintf(buf, "%s <- %s, %d, %d", show_pseudo(insn->target
447                 break;

449         case OP_NOT: case OP_NEG:
450                 buf += sprintf(buf, "%s <- %s", show_pseudo(insn->target), show_
451                 break;

453         case OP_CONTEXT:
454                 buf += sprintf(buf, "%s%d", insn->check ? "check: " : "", insn->
455                 break;
456         case OP_RANGE:
```



```
457                 buf += sprintf(buf, "%s between %s..%s", show_pseudo(insn->src1)
458                 break;
459         case OP_NOP:
460                 buf += sprintf(buf, "%s <- %s", show_pseudo(insn->target), show_
461                 break;
462         case OP_DEATHNOTE:
463                 buf += sprintf(buf, "%s", show_pseudo(insn->target));
464                 break;
465         case OP_ASM:
466                 buf = show_asm(buf, insn);
467                 break;
468         case OP_COPY:
469                 buf += sprintf(buf, "%s <- %s", show_pseudo(insn->target), show_
470                 break;
471         default:
472                 break;
473         }

475         if (buf >= buffer + sizeof(buffer))
476                 die("instruction buffer overflowed %td\n", buf - buffer);
477         do { --buf; } while (*buf == ' ');
478         *++buf = 0;
479         return buffer;
480 }

482 void show_bb(struct basic_block *bb)
483 {
484         struct instruction *insn;

486         printf(".L%u:\n", bb->nr);
487         if (verbose) {
488                 pseudo_t needs, defines;
489                 printf("%s:%d\n", stream_name(bb->pos.stream), bb->pos.line);

491                 FOR_EACH_PTR(bb->needs, needs) {
492                         struct instruction *def = needs->def;
493                         if (def->opcode != OP_PHI) {
494                                 printf("  **uses %s (from .L%u)**\n", show_pseud
495                         } else {
496                                 pseudo_t phi;
497                                 const char *sep = " ";
498                                 printf("  **uses %s (from", show_pseudo(needs));
499                                 FOR_EACH_PTR(def->phi_list, phi) {
500                                         if (phi == VOID)
501                                                 continue;
502                                         printf("%s(%s:.L%u)", sep, show_pseudo(p
503                                         sep = ", ";
504                                 } END_FOR_EACH_PTR(phi);
505                                 printf(")**\n");
506                         }
507                 } END_FOR_EACH_PTR(needs);

509                 FOR_EACH_PTR(bb->defines, defines) {
510                         printf("  **defines %s **\n", show_pseudo(defines));
511                 } END_FOR_EACH_PTR(defines);

513                 if (bb->parents) {
514                         struct basic_block *from;
515                         FOR_EACH_PTR(bb->parents, from) {
516                                 printf("  **from .L%u (%s:%d:%d)**\n", from->nr,
517                                         stream_name(from->pos.stream), from->pos
518                         } END_FOR_EACH_PTR(from);
519                 }

521                 if (bb->children) {
522                         struct basic_block *to;
```

```
523                         FOR_EACH_PTR(bb->children, to) {
524                                 printf("  **to .L%u (%s:%d:%d)**\n", to->nr,
525                                         stream_name(to->pos.stream), to->pos.lin
526                         } END_FOR_EACH_PTR(to);
527                 }
528         }

530         FOR_EACH_PTR(bb->insns, insn) {
531                 if (!insn->bb && verbose < 2)
532                         continue;
533                 printf("\t%s\n", show_instruction(insn));
534         } END_FOR_EACH_PTR(insn);
535         if (!bb_terminated(bb))
536                 printf("\tEND\n");
537 }

539 static void show_symbol_usage(pseudo_t pseudo)
540 {
541         struct pseudo_user *pu;

543         if (pseudo) {
544                 FOR_EACH_PTR(pseudo->users, pu) {
545                         printf("\t%s\n", show_instruction(pu->insn));
546                 } END_FOR_EACH_PTR(pu);
547         }
548 }

550 void show_entry(struct entrypoint *ep)
551 {
552         struct symbol *sym;
553         struct basic_block *bb;

555         printf("%s:\n", show_ident(ep->name->ident));

557         if (verbose) {
558                 printf("ep %p: %s\n", ep, show_ident(ep->name->ident));

560                 FOR_EACH_PTR(ep->syms, sym) {
561                         if (!sym->pseudo)
562                                 continue;
563                         if (!sym->pseudo->users)
564                                 continue;
565                         printf("   sym: %p %s\n", sym, show_ident(sym->ident));
566                         if (sym->ctype.modifiers & (MOD_EXTERN | MOD_STATIC | MO
567                                 printf("\texternal visibility\n");
568                         show_symbol_usage(sym->pseudo);
569                 } END_FOR_EACH_PTR(sym);

571                 printf("\n");
572         }

574         FOR_EACH_PTR(ep->bbs, bb) {
575                 if (!bb)
576                         continue;
577                 if (!bb->parents && !bb->children && !bb->insns && verbose < 2)
578                         continue;
579                 show_bb(bb);
580                 printf("\n");
581         } END_FOR_EACH_PTR(bb);

583         printf("\n");
584 }

586 static void bind_label(struct symbol *label, struct basic_block *bb, struct posi
587 {
588         if (label->bb_target)
```



```
589                 warning(pos, "label '%s' already bound", show_ident(label->ident
590         label->bb_target = bb;
591 }

593 static struct basic_block * get_bound_block(struct entrypoint *ep, struct symbol
594 {
595         struct basic_block *bb = label->bb_target;

597         if (!bb) {
598                 bb = alloc_basic_block(ep, label->pos);
599                 label->bb_target = bb;
600         }
601         return bb;
602 }

604 static void finish_block(struct entrypoint *ep)
605 {
606         struct basic_block *src = ep->active;
607         if (bb_reachable(src))
608                 ep->active = NULL;
609 }

611 static void add_goto(struct entrypoint *ep, struct basic_block *dst)
612 {
613         struct basic_block *src = ep->active;
614         if (bb_reachable(src)) {
615                 struct instruction *br = alloc_instruction(OP_BR, 0);
616                 br->bb_true = dst;
617                 add_bb(&dst->parents, src);
618                 add_bb(&src->children, dst);
619                 br->bb = src;
620                 add_instruction(&src->insns, br);
621                 ep->active = NULL;
622         }
623 }

625 static void add_one_insn(struct entrypoint *ep, struct instruction *insn)
626 {
627         struct basic_block *bb = ep->active;

629         if (bb_reachable(bb)) {
630                 insn->bb = bb;
631                 add_instruction(&bb->insns, insn);
632         }
633 }

635 static void set_activeblock(struct entrypoint *ep, struct basic_block *bb)
636 {
637         if (!bb_terminated(ep->active))
638                 add_goto(ep, bb);

640         ep->active = bb;
641         if (bb_reachable(bb))
642                 add_bb(&ep->bbs, bb);
643 }

645 static void remove_parent(struct basic_block *child, struct basic_block *parent)
646 {
647         remove_bb_from_list(&child->parents, parent, 1);
648         if (!child->parents)
649                 repeat_phase |= REPEAT_CFG_CLEANUP;
650 }

652 /* Change a "switch" or a conditional branch into a branch */
653 void insert_branch(struct basic_block *bb, struct instruction *jmp, struct basic
654 {
```

```
655         struct instruction *br, *old;
656         struct basic_block *child;

658         /* Remove the switch */
659         old = delete_last_instruction(&bb->insns);
660         assert(old == jmp);
661         kill_instruction(old);

663         br = alloc_instruction(OP_BR, 0);
664         br->bb = bb;
665         br->bb_true = target;
666         add_instruction(&bb->insns, br);

668         FOR_EACH_PTR(bb->children, child) {
669                 if (child == target) {
670                         target = NULL;  /* Trigger just once */
671                         continue;
672                 }
673                 DELETE_CURRENT_PTR(child);
674                 remove_parent(child, bb);
675         } END_FOR_EACH_PTR(child);
676         PACK_PTR_LIST(&bb->children);
677 }
678

680 void insert_select(struct basic_block *bb, struct instruction *br, struct instru
681 {
682         pseudo_t target;
683         struct instruction *select;

685         /* Remove the 'br' */
686         delete_last_instruction(&bb->insns);

688         select = alloc_instruction(OP_SEL, phi_node->size);
689         select->bb = bb;

691         assert(br->cond);
692         use_pseudo(select, br->cond, &select->src1);

694         target = phi_node->target;
695         assert(target->def == phi_node);
696         select->target = target;
697         target->def = select;

699         use_pseudo(select, if_true, &select->src2);
700         use_pseudo(select, if_false, &select->src3);

702         add_instruction(&bb->insns, select);
703         add_instruction(&bb->insns, br);
704 }

706 static inline int bb_empty(struct basic_block *bb)
707 {
708         return !bb->insns;
709 }

711 /* Add a label to the currently active block, return new active block */
712 static struct basic_block * add_label(struct entrypoint *ep, struct symbol *labe
713 {
714         struct basic_block *bb = label->bb_target;

716         if (bb) {
717                 set_activeblock(ep, bb);
718                 return bb;
719         }
720         bb = ep->active;
```



```
721         if (!bb_reachable(bb) || !bb_empty(bb)) {
722                 bb = alloc_basic_block(ep, label->pos);
723                 set_activeblock(ep, bb);
724         }
725         label->bb_target = bb;
726         return bb;
727 }

729 static void add_branch(struct entrypoint *ep, struct expression *expr, pseudo_t
730 {
731         struct basic_block *bb = ep->active;
732         struct instruction *br;

734         if (bb_reachable(bb)) {
735                 br = alloc_instruction(OP_CBR, 0);
736                 use_pseudo(br, cond, &br->cond);
737                 br->bb_true = bb_true;
738                 br->bb_false = bb_false;
739                 add_bb(&bb_true->parents, bb);
740                 add_bb(&bb_false->parents, bb);
741                 add_bb(&bb->children, bb_true);
742                 add_bb(&bb->children, bb_false);
743                 add_one_insn(ep, br);
744         }
745 }

747 /* Dummy pseudo allocator */
748 pseudo_t alloc_pseudo(struct instruction *def)
749 {
750         static int nr = 0;
751         struct pseudo * pseudo = __alloc_pseudo(0);
752         pseudo->type = PSEUDO_REG;
753         pseudo->nr = ++nr;
754         pseudo->def = def;
755         return pseudo;
756 }

758 static void clear_symbol_pseudos(struct entrypoint *ep)
759 {
760         pseudo_t pseudo;

762         FOR_EACH_PTR(ep->accesses, pseudo) {
763                 pseudo->sym->pseudo = NULL;
764         } END_FOR_EACH_PTR(pseudo);
765 }

767 static pseudo_t symbol_pseudo(struct entrypoint *ep, struct symbol *sym)
768 {
769         pseudo_t pseudo;

771         if (!sym)
772                 return VOID;

774         pseudo = sym->pseudo;
775         if (!pseudo) {
776                 pseudo = __alloc_pseudo(0);
777                 pseudo->nr = -1;
778                 pseudo->type = PSEUDO_SYM;
779                 pseudo->sym = sym;
780                 pseudo->ident = sym->ident;
781                 sym->pseudo = pseudo;
782                 add_pseudo(&ep->accesses, pseudo);
783         }
784         /* Symbol pseudos have neither nr, usage nor def */
785         return pseudo;
786 }
```

```
pseudo_t value_pseudo(struct symbol *type, long long val)
{
#define MAX_VAL_HASH 64
        static struct pseudo_list *prev[MAX_VAL_HASH];
        int hash = val & (MAX_VAL_HASH-1);
        struct pseudo_list **list = prev + hash;
        int size = type ? type->bit_size : value_size(val);
        pseudo_t pseudo;


        FOR_EACH_PTR(*list, pseudo) {
                if (pseudo->value == val && pseudo->size == size)
                        return pseudo;
        } END_FOR_EACH_PTR(pseudo);

        pseudo = __alloc_pseudo(0);
        pseudo->type = PSEUDO_VAL;
        pseudo->value = val;
        pseudo->size = size;
        add_pseudo(list, pseudo);

        /* Value pseudos have neither nr, usage nor def */
        return pseudo;
}

static pseudo_t argument_pseudo(struct entrypoint *ep, int nr)
{
        pseudo_t pseudo = __alloc_pseudo(0);
        struct instruction *entry = ep->entry;

        pseudo->type = PSEUDO_ARG;
        pseudo->nr = nr;
        pseudo->def = entry;
        add_pseudo(&entry->arg_list, pseudo);

        /* Argument pseudos have neither usage nor def */
        return pseudo;
}

pseudo_t alloc_phi(struct basic_block *source, pseudo_t pseudo, int size)
{
        struct instruction *insn;
        pseudo_t phi;
        static int nr = 0;

        if (!source)
                return VOID;

        insn = alloc_instruction(OP_PHISOURCE, size);
        phi = __alloc_pseudo(0);
        phi->type = PSEUDO_PHI;
        phi->nr = ++nr;
        phi->def = insn;

        use_pseudo(insn, pseudo, &insn->phi_src);
        insn->bb = source;
        insn->target = phi;
        add_instruction(&source->insns, insn);
        return phi;
}

/*
 * We carry the "access_data" structure around for any accesses,
 * which simplifies things a lot. It contains all the access
 * information in one place.
```



```
 */
struct access_data {
        struct symbol *result_type;     // result ctype
        struct symbol *source_type;     // source ctype
        pseudo_t address;               // pseudo containing address ..
        unsigned int offset;            // byte offset
        struct position pos;
};

static void finish_address_gen(struct entrypoint *ep, struct access_data *ad)
{
}

static int linearize_simple_address(struct entrypoint *ep,
        struct expression *addr,
        struct access_data *ad)
{
        if (addr->type == EXPR_SYMBOL) {
                linearize_one_symbol(ep, addr->symbol);
                ad->address = symbol_pseudo(ep, addr->symbol);
                return 1;
        }
        if (addr->type == EXPR_BINOP) {
                if (addr->right->type == EXPR_VALUE) {
                        if (addr->op == '+') {
                                ad->offset += get_expression_value(addr->right);
                                return linearize_simple_address(ep, addr->left,
                        }
                }
        }
        ad->address = linearize_expression(ep, addr);
        return 1;
}

static struct symbol *base_type(struct symbol *sym)
{
        struct symbol *base = sym;

        if (sym) {
                if (sym->type == SYM_NODE)
                        base = base->ctype.base_type;
                if (base->type == SYM_BITFIELD)
                        return base->ctype.base_type;
        }
        return sym;
}

static int linearize_address_gen(struct entrypoint *ep,
        struct expression *expr,
        struct access_data *ad)
{
        struct symbol *ctype = expr->ctype;

        if (!ctype)
                return 0;
        ad->pos = expr->pos;
        ad->result_type = ctype;
        ad->source_type = base_type(ctype);
        if (expr->type == EXPR_PREOP && expr->op == '*')
                return linearize_simple_address(ep, expr->unop, ad);

        warning(expr->pos, "generating address of non-lvalue (%d)", expr->type);
        return 0;
}

static pseudo_t add_load(struct entrypoint *ep, struct access_data *ad)
```

```
919 {
920         struct instruction *insn;
921         pseudo_t new;

923         insn = alloc_typed_instruction(OP_LOAD, ad->source_type);
924         new = alloc_pseudo(insn);

926         insn->target = new;
927         insn->offset = ad->offset;
928         use_pseudo(insn, ad->address, &insn->src);
929         add_one_insn(ep, insn);
930         return new;
931 }

933 static void add_store(struct entrypoint *ep, struct access_data *ad, pseudo_t va
934 {
935         struct basic_block *bb = ep->active;

937         if (bb_reachable(bb)) {
938                 struct instruction *store = alloc_typed_instruction(OP_STORE, ad
939                 store->offset = ad->offset;
940                 use_pseudo(store, value, &store->target);
941                 use_pseudo(store, ad->address, &store->src);
942                 add_one_insn(ep, store);
943         }
944 }

946 static pseudo_t linearize_store_gen(struct entrypoint *ep,
947                 pseudo_t value,
948                 struct access_data *ad)
949 {
950         pseudo_t store = value;

952         if (type_size(ad->source_type) != type_size(ad->result_type)) {
953                 struct symbol *ctype = ad->result_type;
954                 unsigned int shift = ctype->bit_offset;
955                 unsigned int size = ctype->bit_size;
956                 pseudo_t orig = add_load(ep, ad);
957                 unsigned long long mask = (1ULL << size) - 1;

959                 if (shift) {
960                         store = add_binary_op(ep, ad->source_type, OP_SHL, value
961                         mask <<= shift;
962                 }
963                 orig = add_binary_op(ep, ad->source_type, OP_AND, orig, value_ps
964                 store = add_binary_op(ep, ad->source_type, OP_OR, orig, store);
965         }
966         add_store(ep, ad, store);
967         return value;
968 }

970 static pseudo_t add_binary_op(struct entrypoint *ep, struct symbol *ctype, int o
971 {
972         struct instruction *insn = alloc_typed_instruction(op, ctype);
973         pseudo_t target = alloc_pseudo(insn);
974         insn->target = target;
975         use_pseudo(insn, left, &insn->src1);
976         use_pseudo(insn, right, &insn->src2);
977         add_one_insn(ep, insn);
978         return target;
979 }

981 static pseudo_t add_setval(struct entrypoint *ep, struct symbol *ctype, struct e
982 {
983         struct instruction *insn = alloc_typed_instruction(OP_SETVAL, ctype);
984         pseudo_t target = alloc_pseudo(insn);
```



```
985         insn->target = target;
986         insn->val = val;
987         add_one_insn(ep, insn);
988         return target;
989 }

991 static pseudo_t add_symbol_address(struct entrypoint *ep, struct symbol *sym)
992 {
993         struct instruction *insn = alloc_instruction(OP_SYMADDR, bits_in_pointer
994         pseudo_t target = alloc_pseudo(insn);

996         insn->target = target;
997         use_pseudo(insn, symbol_pseudo(ep, sym), &insn->symbol);
998         add_one_insn(ep, insn);
999         return target;
1000 }

1002 static pseudo_t linearize_load_gen(struct entrypoint *ep, struct access_data *ad
1003 {
1004         struct symbol *ctype = ad->result_type;
1005         pseudo_t new = add_load(ep, ad);

1007         if (ctype->bit_offset) {
1008                 pseudo_t shift = value_pseudo(ctype, ctype->bit_offset);
1009                 pseudo_t newval = add_binary_op(ep, ad->source_type, OP_LSR, new
1010                 new = newval;
1011         }
1012         if (ctype->bit_size != type_size(ad->source_type))
1013                 new = cast_pseudo(ep, new, ad->source_type, ad->result_type);
1014         return new;
1015 }

1017 static pseudo_t linearize_access(struct entrypoint *ep, struct expression *expr)
1018 {
1019         struct access_data ad = { NULL, };
1020         pseudo_t value;

1022         if (!linearize_address_gen(ep, expr, &ad))
1023                 return VOID;
1024         value = linearize_load_gen(ep, &ad);
1025         finish_address_gen(ep, &ad);
1026         return value;
1027 }

1029 /* FIXME: FP */
1030 static pseudo_t linearize_inc_dec(struct entrypoint *ep, struct expression *expr
1031 {
1032         struct access_data ad = { NULL, };
1033                 pseudo_t old, new, one;
1034         int op = expr->op == SPECIAL_INCREMENT ? OP_ADD : OP_SUB;

1036         if (!linearize_address_gen(ep, expr->unop, &ad))
1037                 return VOID;

1039         old = linearize_load_gen(ep, &ad);
1040         one = value_pseudo(expr->ctype, expr->op_value);
1041         new = add_binary_op(ep, expr->ctype, op, old, one);
1042         linearize_store_gen(ep, new, &ad);
1043         finish_address_gen(ep, &ad);
1044         return postop ? old : new;
1045 }

1047 static pseudo_t add_uniop(struct entrypoint *ep, struct expression *expr, int op
1048 {
1049         struct instruction *insn = alloc_typed_instruction(op, expr->ctype);
1050         pseudo_t new = alloc_pseudo(insn);
```

```
1052         insn->target = new;
1053         use_pseudo(insn, src, &insn->src1);
1054         add_one_insn(ep, insn);
1055         return new;
1056 }

1058 static pseudo_t linearize_slice(struct entrypoint *ep, struct expression *expr)
1059 {
1060         pseudo_t pre = linearize_expression(ep, expr->base);
1061         struct instruction *insn = alloc_typed_instruction(OP_SLICE, expr->ctype
1062         pseudo_t new = alloc_pseudo(insn);

1064         insn->target = new;
1065         insn->from = expr->r_bitpos;
1066         insn->len = expr->r_nrbits;
1067         use_pseudo(insn, pre, &insn->base);
1068         add_one_insn(ep, insn);
1069         return new;
1070 }

1072 static pseudo_t linearize_regular_preop(struct entrypoint *ep, struct expression
1073 {
1074         pseudo_t pre = linearize_expression(ep, expr->unop);
1075         switch (expr->op) {
1076         case '+':
1077                 return pre;
1078         case '!': {
1079                 pseudo_t zero = value_pseudo(expr->ctype, 0);
1080                 return add_binary_op(ep, expr->ctype, OP_SET_EQ, pre, zero);
1081         }
1082         case '~':
1083                 return add_uniop(ep, expr, OP_NOT, pre);
1084         case '-':
1085                 return add_uniop(ep, expr, OP_NEG, pre);
1086         }
1087         return VOID;
1088 }

1090 static pseudo_t linearize_preop(struct entrypoint *ep, struct expression *expr)
1091 {
1092         /*
1093          * '*' is an lvalue access, and is fundamentally different
1094          * from an arithmetic operation. Maybe it should have an
1095          * expression type of its own..
1096          */
1097         if (expr->op == '*')
1098                 return linearize_access(ep, expr);
1099         if (expr->op == SPECIAL_INCREMENT || expr->op == SPECIAL_DECREMENT)
1100                 return linearize_inc_dec(ep, expr, 0);
1101         return linearize_regular_preop(ep, expr);
1102 }

1104 static pseudo_t linearize_postop(struct entrypoint *ep, struct expression *expr)
1105 {
1106         return linearize_inc_dec(ep, expr, 1);
1107 }

1109 /*
1110  * Casts to pointers are "less safe" than other casts, since
1111  * they imply type-unsafe accesses. "void *" is a special
1112  * case, since you can't access through it anyway without another
1113  * cast.
1114  */
1115 static struct instruction *alloc_cast_instruction(struct symbol *src, struct sym
1116 {
```



```
1117         int opcode = OP_CAST;
1118         struct symbol *base = ctype;

1120         if (src->ctype.modifiers & MOD_SIGNED)
1121                 opcode = OP_SCAST;
1122         if (base->type == SYM_NODE)
1123                 base = base->ctype.base_type;
1124         if (base->type == SYM_PTR) {
1125                 base = base->ctype.base_type;
1126                 if (base != &void_ctype)
1127                         opcode = OP_PTRCAST;
1128         } else if (base->ctype.base_type == &fp_type)
1129                 opcode = OP_FPCAST;
1130         return alloc_typed_instruction(opcode, ctype);
1131 }

1133 static pseudo_t cast_pseudo(struct entrypoint *ep, pseudo_t src, struct symbol *
1134 {
1135         pseudo_t result;
1136         struct instruction *insn;

1138         if (src == VOID)
1139                 return VOID;
1140         if (!from || !to)
1141                 return VOID;
1142         if (from->bit_size < 0 || to->bit_size < 0)
1143                 return VOID;
1144         insn = alloc_cast_instruction(from, to);
1145         result = alloc_pseudo(insn);
1146         insn->target = result;
1147         insn->orig_type = from;
1148         use_pseudo(insn, src, &insn->src);
1149         add_one_insn(ep, insn);
1150         return result;
1151 }

1153 static int opcode_sign(int opcode, struct symbol *ctype)
1154 {
1155         if (ctype && (ctype->ctype.modifiers & MOD_SIGNED)) {
1156                 switch(opcode) {
1157                 case OP_MULU: case OP_DIVU: case OP_MODU: case OP_LSR:
1158                         opcode++;
1159                 }
1160         }
1161         return opcode;
1162 }

1164 static inline pseudo_t add_convert_to_bool(struct entrypoint *ep, pseudo_t src,
1165 {
1166         pseudo_t zero;
1167         int op;

1169         if (is_bool_type(type))
1170                 return src;
1171         zero = value_pseudo(type, 0);
1172         op = OP_SET_NE;
1173         return add_binary_op(ep, &bool_ctype, op, src, zero);
1174 }

1176 static pseudo_t linearize_expression_to_bool(struct entrypoint *ep, struct expre
1177 {
1178         pseudo_t dst;
1179         dst = linearize_expression(ep, expr);
1180         dst = add_convert_to_bool(ep, dst, expr->ctype);
1181         return dst;
1182 }
```

```
1184 static pseudo_t linearize_assignment(struct entrypoint *ep, struct expression *e
1185 {
1186         struct access_data ad = { NULL, };
1187         struct expression *target = expr->left;
1188         struct expression *src = expr->right;
1189         struct symbol *ctype;
1190         pseudo_t value;
1191
1192         value = linearize_expression(ep, src);
1193         if (!target || !linearize_address_gen(ep, target, &ad))
1194                 return value;
1195         if (expr->op != '=') {
1196                 pseudo_t oldvalue = linearize_load_gen(ep, &ad);
1197                 pseudo_t dst;
1198                 static const int op_trans[] = {
1199                         [SPECIAL_ADD_ASSIGN - SPECIAL_BASE] = OP_ADD,
1200                         [SPECIAL_SUB_ASSIGN - SPECIAL_BASE] = OP_SUB,
1201                         [SPECIAL_MUL_ASSIGN - SPECIAL_BASE] = OP_MULU,
1202                         [SPECIAL_DIV_ASSIGN - SPECIAL_BASE] = OP_DIVU,
1203                         [SPECIAL_MOD_ASSIGN - SPECIAL_BASE] = OP_MODU,
1204                         [SPECIAL_SHL_ASSIGN - SPECIAL_BASE] = OP_SHL,
1205                         [SPECIAL_SHR_ASSIGN - SPECIAL_BASE] = OP_LSR,
1206                         [SPECIAL_AND_ASSIGN - SPECIAL_BASE] = OP_AND,
1207                         [SPECIAL_OR_ASSIGN  - SPECIAL_BASE] = OP_OR,
1208                         [SPECIAL_XOR_ASSIGN - SPECIAL_BASE] = OP_XOR
1209                 };
1210                 int opcode;
1211
1212                 if (!src)
1213                         return VOID;
1214
1215                 ctype = src->ctype;
1216                 oldvalue = cast_pseudo(ep, oldvalue, target->ctype, ctype);
1217                 opcode = opcode_sign(op_trans[expr->op - SPECIAL_BASE], ctype);
1218                 dst = add_binary_op(ep, ctype, opcode, oldvalue, value);
1219                 value = cast_pseudo(ep, dst, ctype, expr->ctype);
1220         }
1221         value = linearize_store_gen(ep, value, &ad);
1222         finish_address_gen(ep, &ad);
1223         return value;
1224 }
1225
1226 static pseudo_t linearize_call_expression(struct entrypoint *ep, struct expressi
1227 {
1228         struct expression *arg, *fn;
1229         struct instruction *insn = alloc_typed_instruction(OP_CALL, expr->ctype)
1230         pseudo_t retval, call;
1231         struct ctype *ctype = NULL;
1232         struct symbol *fntype;
1233         struct context *context;
1234
1235         if (!expr->ctype) {
1236                 warning(expr->pos, "call with no type!");
1237                 return VOID;
1238         }
1239
1240         FOR_EACH_PTR(expr->args, arg) {
1241                 pseudo_t new = linearize_expression(ep, arg);
1242                 use_pseudo(insn, new, add_pseudo(&insn->arguments, new));
1243         } END_FOR_EACH_PTR(arg);
1244
1245         fn = expr->fn;
1246
1247         if (fn->ctype)
1248                 ctype = &fn->ctype->ctype;
```



```
1250         fntype = fn->ctype;
1251         if (fntype) {
1252                 if (fntype->type == SYM_NODE)
1253                         fntype = fntype->ctype.base_type;
1254         }
1255         insn->fntype = fntype;
1256
1257         if (fn->type == EXPR_PREOP) {
1258                 if (fn->unop->type == EXPR_SYMBOL) {
1259                         struct symbol *sym = fn->unop->symbol;
1260                         if (sym->ctype.base_type->type == SYM_FN)
1261                                 fn = fn->unop;
1262                 }
1263         }
1264         if (fn->type == EXPR_SYMBOL) {
1265                 call = symbol_pseudo(ep, fn->symbol);
1266         } else {
1267                 call = linearize_expression(ep, fn);
1268         }
1269         use_pseudo(insn, call, &insn->func);
1270         retval = VOID;
1271         if (expr->ctype != &void_ctype)
1272                 retval = alloc_pseudo(insn);
1273         insn->target = retval;
1274         add_one_insn(ep, insn);
1275
1276         if (ctype) {
1277                 FOR_EACH_PTR(ctype->contexts, context) {
1278                         int in = context->in;
1279                         int out = context->out;
1280                         int check = 0;
1281                         int context_diff;
1282                         if (in < 0) {
1283                                 check = 1;
1284                                 in = 0;
1285                         }
1286                         if (out < 0) {
1287                                 check = 0;
1288                                 out = 0;
1289                         }
1290                         context_diff = out - in;
1291                         if (check || context_diff) {
1292                                 insn = alloc_instruction(OP_CONTEXT, 0);
1293                                 insn->increment = context_diff;
1294                                 insn->check = check;
1295                                 insn->context_expr = context->context;
1296                                 add_one_insn(ep, insn);
1297                         }
1298                 } END_FOR_EACH_PTR(context);
1299         }
1300
1301         return retval;
1302 }
1303
1304 static pseudo_t linearize_binop_bool(struct entrypoint *ep, struct expression *e
1305 {
1306         pseudo_t src1, src2, dst;
1307         int op = (expr->op == SPECIAL_LOGICAL_OR) ? OP_OR_BOOL : OP_AND_BOOL;
1308
1309         src1 = linearize_expression_to_bool(ep, expr->left);
1310         src2 = linearize_expression_to_bool(ep, expr->right);
1311         dst = add_binary_op(ep, &bool_ctype, op, src1, src2);
1312         if (expr->ctype != &bool_ctype)
1313                 dst = cast_pseudo(ep, dst, &bool_ctype, expr->ctype);
1314         return dst;
```

```
}

static pseudo_t linearize_binop(struct entrypoint *ep, struct expression *expr)
{
        pseudo_t src1, src2, dst;
        static const int opcode[] = {
                ['+'] = OP_ADD, ['-'] = OP_SUB,
                ['*'] = OP_MULU, ['/'] = OP_DIVU,
                ['%'] = OP_MODU, ['&'] = OP_AND,
                ['|'] = OP_OR,  ['^'] = OP_XOR,
                [SPECIAL_LEFTSHIFT] = OP_SHL,
                [SPECIAL_RIGHTSHIFT] = OP_LSR,
        };
        int op;

        src1 = linearize_expression(ep, expr->left);
        src2 = linearize_expression(ep, expr->right);
        op = opcode_sign(opcode[expr->op], expr->ctype);
        dst = add_binary_op(ep, expr->ctype, op, src1, src2);
        return dst;
}

static pseudo_t linearize_logical_branch(struct entrypoint *ep, struct expressio

pseudo_t linearize_cond_branch(struct entrypoint *ep, struct expression *expr, s

static pseudo_t linearize_select(struct entrypoint *ep, struct expression *expr)
{
        pseudo_t cond, true, false, res;
        struct instruction *insn;

        true = linearize_expression(ep, expr->cond_true);
        false = linearize_expression(ep, expr->cond_false);
        cond = linearize_expression(ep, expr->conditional);

        insn = alloc_typed_instruction(OP_SEL, expr->ctype);
        if (!expr->cond_true)
                true = cond;
        use_pseudo(insn, cond, &insn->src1);
        use_pseudo(insn, true, &insn->src2);
        use_pseudo(insn, false, &insn->src3);

        res = alloc_pseudo(insn);
        insn->target = res;
        add_one_insn(ep, insn);
        return res;
}

static pseudo_t add_join_conditional(struct entrypoint *ep, struct expression *e
                                     pseudo_t phi1, pseudo_t phi2)
{
        pseudo_t target;
        struct instruction *phi_node;

        if (phi1 == VOID)
                return phi2;
        if (phi2 == VOID)
                return phi1;

        phi_node = alloc_typed_instruction(OP_PHI, expr->ctype);
        use_pseudo(phi_node, phi1, add_pseudo(&phi_node->phi_list, phi1));
        use_pseudo(phi_node, phi2, add_pseudo(&phi_node->phi_list, phi2));
        phi_node->target = target = alloc_pseudo(phi_node);
        add_one_insn(ep, phi_node);
        return target;
}
```



```
static pseudo_t linearize_short_conditional(struct entrypoint *ep, struct expres
                                            struct expression *cond,
                                            struct expression *expr_false)
{
        pseudo_t src1, src2;
        struct basic_block *bb_false;
        struct basic_block *merge = alloc_basic_block(ep, expr->pos);
        pseudo_t phi1, phi2;
        int size = type_size(expr->ctype);

        if (!expr_false || !ep->active)
                return VOID;

        bb_false = alloc_basic_block(ep, expr_false->pos);
        src1 = linearize_expression(ep, cond);
        phi1 = alloc_phi(ep->active, src1, size);
        add_branch(ep, expr, src1, merge, bb_false);

        set_activeblock(ep, bb_false);
        src2 = linearize_expression(ep, expr_false);
        phi2 = alloc_phi(ep->active, src2, size);
        set_activeblock(ep, merge);

        return add_join_conditional(ep, expr, phi1, phi2);
}

static pseudo_t linearize_conditional(struct entrypoint *ep, struct expression *
                                      struct expression *cond,
                                      struct expression *expr_true,
                                      struct expression *expr_false)
{
        pseudo_t src1, src2;
        pseudo_t phi1, phi2;
        struct basic_block *bb_true, *bb_false, *merge;
        int size = type_size(expr->ctype);

        if (!cond || !expr_true || !expr_false || !ep->active)
                return VOID;
        bb_true = alloc_basic_block(ep, expr_true->pos);
        bb_false = alloc_basic_block(ep, expr_false->pos);
        merge = alloc_basic_block(ep, expr->pos);

        linearize_cond_branch(ep, cond, bb_true, bb_false);

        set_activeblock(ep, bb_true);
        src1 = linearize_expression(ep, expr_true);
        phi1 = alloc_phi(ep->active, src1, size);
        add_goto(ep, merge);

        set_activeblock(ep, bb_false);
        src2 = linearize_expression(ep, expr_false);
        phi2 = alloc_phi(ep->active, src2, size);
        set_activeblock(ep, merge);

        return add_join_conditional(ep, expr, phi1, phi2);
}

static pseudo_t linearize_logical(struct entrypoint *ep, struct expression *expr
{
        struct expression *shortcut;

        shortcut = alloc_const_expression(expr->pos, expr->op == SPECIAL_LOGICAL
        shortcut->ctype = expr->ctype;
        if (expr->op == SPECIAL_LOGICAL_OR)
                return linearize_conditional(ep, expr, expr->left, shortcut, exp
```

```
1447         return linearize_conditional(ep, expr, expr->left, expr->right, shortcut
1448 }

1450 static pseudo_t linearize_compare(struct entrypoint *ep, struct expression *expr
1451 {
1452         static const int cmpop[] = {
1453                 ['>'] = OP_SET_GT, ['<'] = OP_SET_LT,
1454                 [SPECIAL_EQUAL] = OP_SET_EQ,
1455                 [SPECIAL_NOTEQUAL] = OP_SET_NE,
1456                 [SPECIAL_GTE] = OP_SET_GE,
1457                 [SPECIAL_LTE] = OP_SET_LE,
1458                 [SPECIAL_UNSIGNED_LT] = OP_SET_B,
1459                 [SPECIAL_UNSIGNED_GT] = OP_SET_A,
1460                 [SPECIAL_UNSIGNED_LTE] = OP_SET_BE,
1461                 [SPECIAL_UNSIGNED_GTE] = OP_SET_AE,
1462         };

1464         pseudo_t src1 = linearize_expression(ep, expr->left);
1465         pseudo_t src2 = linearize_expression(ep, expr->right);
1466         pseudo_t dst = add_binary_op(ep, expr->ctype, cmpop[expr->op], src1, src
1467         return dst;
1468 }


1471 pseudo_t linearize_cond_branch(struct entrypoint *ep, struct expression *expr, s
1472 {
1473         pseudo_t cond;

1475         if (!expr || !bb_reachable(ep->active))
1476                 return VOID;

1478         switch (expr->type) {

1480         case EXPR_STRING:
1481         case EXPR_VALUE:
1482                 add_goto(ep, expr->value ? bb_true : bb_false);
1483                 return VOID;

1485         case EXPR_FVALUE:
1486                 add_goto(ep, expr->fvalue ? bb_true : bb_false);
1487                 return VOID;
1488
1489         case EXPR_LOGICAL:
1490                 linearize_logical_branch(ep, expr, bb_true, bb_false);
1491                 return VOID;

1493         case EXPR_COMPARE:
1494                 cond = linearize_compare(ep, expr);
1495                 add_branch(ep, expr, cond, bb_true, bb_false);
1496                 break;
1497
1498         case EXPR_PREOP:
1499                 if (expr->op == '!')
1500                         return linearize_cond_branch(ep, expr->unop, bb_false, b
1501                 /* fall through */
1502         default: {
1503                 cond = linearize_expression(ep, expr);
1504                 add_branch(ep, expr, cond, bb_true, bb_false);

1506                 return VOID;
1507         }
1508         }
1509         return VOID;
1510 }
```



```
1513
1514 static pseudo_t linearize_logical_branch(struct entrypoint *ep, struct expressio
1515 {
1516         struct basic_block *next = alloc_basic_block(ep, expr->pos);

1518         if (expr->op == SPECIAL_LOGICAL_OR)
1519                 linearize_cond_branch(ep, expr->left, bb_true, next);
1520         else
1521                 linearize_cond_branch(ep, expr->left, next, bb_false);
1522         set_activeblock(ep, next);
1523         linearize_cond_branch(ep, expr->right, bb_true, bb_false);
1524         return VOID;
1525 }

1527 static pseudo_t linearize_cast(struct entrypoint *ep, struct expression *expr)
1528 {
1529         pseudo_t src;
1530         struct expression *orig = expr->cast_expression;

1532         if (!orig)
1533                 return VOID;

1535         src = linearize_expression(ep, orig);
1536         return cast_pseudo(ep, src, orig->ctype, expr->ctype);
1537 }

1539 static pseudo_t linearize_position(struct entrypoint *ep, struct expression *pos
1540 {
1541         struct expression *init_expr = pos->init_expr;

1543         ad->offset = pos->init_offset;
1544         ad->source_type = base_type(init_expr->ctype);
1545         ad->result_type = init_expr->ctype;
1546         return linearize_initializer(ep, init_expr, ad);
1547 }

1549 static pseudo_t linearize_initializer(struct entrypoint *ep, struct expression *
1550 {
1551         switch (initializer->type) {
1552         case EXPR_INITIALIZER: {
1553                 struct expression *expr;
1554                 FOR_EACH_PTR(initializer->expr_list, expr) {
1555                         linearize_initializer(ep, expr, ad);
1556                 } END_FOR_EACH_PTR(expr);
1557                 break;
1558         }
1559         case EXPR_POS:
1560                 linearize_position(ep, initializer, ad);
1561                 break;
1562         default: {
1563                 pseudo_t value = linearize_expression(ep, initializer);
1564                 ad->source_type = base_type(initializer->ctype);
1565                 ad->result_type = initializer->ctype;
1566                 linearize_store_gen(ep, value, ad);
1567                 return value;
1568         }
1569         }

1571         return VOID;
1572 }

1574 static void linearize_argument(struct entrypoint *ep, struct symbol *arg, int nr
1575 {
1576         struct access_data ad = { NULL, };

1578         ad.source_type = arg;
```

```
1579         ad.result_type = arg;
1580         ad.address = symbol_pseudo(ep, arg);
1581         linearize_store_gen(ep, argument_pseudo(ep, nr), &ad);
1582         finish_address_gen(ep, &ad);
1583 }

1585 pseudo_t linearize_expression(struct entrypoint *ep, struct expression *expr)
1586 {
1587         if (!expr)
1588                 return VOID;

1590         current_pos = expr->pos;
1591         switch (expr->type) {
1592         case EXPR_SYMBOL:
1593                 linearize_one_symbol(ep, expr->symbol);
1594                 return add_symbol_address(ep, expr->symbol);

1596         case EXPR_VALUE:
1597                 return value_pseudo(expr->ctype, expr->value);

1599         case EXPR_STRING: case EXPR_FVALUE: case EXPR_LABEL:
1600                 return add_setval(ep, expr->ctype, expr);

1602         case EXPR_STATEMENT:
1603                 return linearize_statement(ep, expr->statement);

1605         case EXPR_CALL:
1606                 return linearize_call_expression(ep, expr);

1608         case EXPR_BINOP:
1609                 if (expr->op == SPECIAL_LOGICAL_AND || expr->op == SPECIAL_LOGIC
1610                         return linearize_binop_bool(ep, expr);
1611                 return linearize_binop(ep, expr);

1613         case EXPR_LOGICAL:
1614                 return linearize_logical(ep, expr);

1616         case EXPR_COMPARE:
1617                 return  linearize_compare(ep, expr);

1619         case EXPR_SELECT:
1620                 return  linearize_select(ep, expr);

1622         case EXPR_CONDITIONAL:
1623                 if (!expr->cond_true)
1624                         return linearize_short_conditional(ep, expr, expr->condi

1626                 return  linearize_conditional(ep, expr, expr->conditional,
1627                                               expr->cond_true, expr->cond_false)

1629         case EXPR_COMMA:
1630                 linearize_expression(ep, expr->left);
1631                 return linearize_expression(ep, expr->right);

1633         case EXPR_ASSIGNMENT:
1634                 return linearize_assignment(ep, expr);

1636         case EXPR_PREOP:
1637                 return linearize_preop(ep, expr);

1639         case EXPR_POSTOP:
1640                 return linearize_postop(ep, expr);

1642         case EXPR_CAST:
1643         case EXPR_FORCE_CAST:
1644         case EXPR_IMPLIED_CAST:
```



```
1645                 return linearize_cast(ep, expr);
1646
1647         case EXPR_SLICE:
1648                 return linearize_slice(ep, expr);

1650         case EXPR_INITIALIZER:
1651         case EXPR_POS:
1652                 warning(expr->pos, "unexpected initializer expression (%d %d)",
1653                 return VOID;
1654         default:
1655                 warning(expr->pos, "unknown expression (%d %d)", expr->type, exp
1656                 return VOID;
1657         }
1658         return VOID;
1659 }

1661 static pseudo_t linearize_one_symbol(struct entrypoint *ep, struct symbol *sym)
1662 {
1663         struct access_data ad = { NULL, };
1664         pseudo_t value;

1666         if (!sym || !sym->initializer || sym->initialized)
1667                 return VOID;

1669         /* We need to output these puppies some day too.. */
1670         if (sym->ctype.modifiers & (MOD_STATIC | MOD_TOPLEVEL))
1671                 return VOID;

1673         sym->initialized = 1;
1674         ad.address = symbol_pseudo(ep, sym);

1676         if (sym->initializer && !is_scalar_type(sym)) {
1677                 // default zero initialization [6.7.9.21]
1678                 // FIXME: this init the whole aggregate while
1679                 // only the existing fields need to be initialized.
1680                 // FIXME: this init the whole aggregate even if
1681                 // all fields arelater  explicitely initialized.
1682                 struct expression *expr = sym->initializer;
1683                 ad.pos = expr->pos;
1684                 ad.result_type = sym;
1685                 ad.source_type = base_type(sym);
1686                 ad.address = symbol_pseudo(ep, sym);
1687                 linearize_store_gen(ep, value_pseudo(sym, 0), &ad);
1688         }

1690         value = linearize_initializer(ep, sym->initializer, &ad);
1691         finish_address_gen(ep, &ad);
1692         return value;
1693 }

1695 static pseudo_t linearize_compound_statement(struct entrypoint *ep, struct state
1696 {
1697         pseudo_t pseudo;
1698         struct statement *s;
1699         struct symbol *ret = stmt->ret;

1701         pseudo = VOID;
1702         FOR_EACH_PTR(stmt->stmts, s) {
1703                 pseudo = linearize_statement(ep, s);
1704         } END_FOR_EACH_PTR(s);

1706         if (ret) {
1707                 struct basic_block *bb = add_label(ep, ret);
1708                 struct instruction *phi_node = first_instruction(bb->insns);

1710                 if (!phi_node)
```

```
1711                        return pseudo;

1713                if (pseudo_list_size(phi_node->phi_list)==1) {
1714                        pseudo = first_pseudo(phi_node->phi_list);
1715                        assert(pseudo->type == PSEUDO_PHI);
1716                        return pseudo->def->src1;
1717                }
1718                return phi_node->target;
1719        }

1721        return pseudo;
1722 }

1724 static pseudo_t linearize_inlined_call(struct entrypoint *ep, struct statement *
1725 {
1726        struct instruction *insn = alloc_instruction(OP_INLINED_CALL, 0);
1727        struct statement *args = stmt->args;
1728        struct basic_block *bb;
1729        pseudo_t pseudo;

1731        if (args) {
1732                struct symbol *sym;

1734                concat_symbol_list(args->declaration, &ep->syms);
1735                FOR_EACH_PTR(args->declaration, sym) {
1736                        pseudo_t value = linearize_one_symbol(ep, sym);
1737                        use_pseudo(insn, value, add_pseudo(&insn->arguments, val
1738                } END_FOR_EACH_PTR(sym);
1739        }

1741        insn->target = pseudo = linearize_compound_statement(ep, stmt);
1742        use_pseudo(insn, symbol_pseudo(ep, stmt->inline_fn), &insn->func);
1743        bb = ep->active;
1744        if (bb && !bb->insns)
1745                bb->pos = stmt->pos;
1746        add_one_insn(ep, insn);
1747        return pseudo;
1748 }

1750 static pseudo_t linearize_context(struct entrypoint *ep, struct statement *stmt)
1751 {
1752        struct instruction *insn = alloc_instruction(OP_CONTEXT, 0);
1753        struct expression *expr = stmt->expression;
1754        int value = 0;

1756        if (expr->type == EXPR_VALUE)
1757                value = expr->value;

1759        insn->increment = value;
1760        insn->context_expr = stmt->context;
1761        add_one_insn(ep, insn);
1762        return VOID;
1763 }

1765 static pseudo_t linearize_range(struct entrypoint *ep, struct statement *stmt)
1766 {
1767        struct instruction *insn = alloc_instruction(OP_RANGE, 0);

1769        use_pseudo(insn, linearize_expression(ep, stmt->range_expression), &insn
1770        use_pseudo(insn, linearize_expression(ep, stmt->range_low), &insn->src2)
1771        use_pseudo(insn, linearize_expression(ep, stmt->range_high), &insn->src3
1772        add_one_insn(ep, insn);
1773        return VOID;
1774 }

1776 ALLOCATOR(asm_rules, "asm rules");
```



```
1777 ALLOCATOR(asm_constraint, "asm constraints");

1779 static void add_asm_input(struct entrypoint *ep, struct instruction *insn, struc
1780        const char *constraint, const struct ident *ident)
1781 {
1782        pseudo_t pseudo = linearize_expression(ep, expr);
1783        struct asm_constraint *rule = __alloc_asm_constraint(0);

1785        rule->ident = ident;
1786        rule->constraint = constraint;
1787        use_pseudo(insn, pseudo, &rule->pseudo);
1788        add_ptr_list(&insn->asm_rules->inputs, rule);
1789 }

1791 static void add_asm_output(struct entrypoint *ep, struct instruction *insn, stru
1792        const char *constraint, const struct ident *ident)
1793 {
1794        struct access_data ad = { NULL, };
1795        pseudo_t pseudo = alloc_pseudo(insn);
1796        struct asm_constraint *rule;

1798        if (!expr || !linearize_address_gen(ep, expr, &ad))
1799                return;
1800        linearize_store_gen(ep, pseudo, &ad);
1801        finish_address_gen(ep, &ad);
1802        rule = __alloc_asm_constraint(0);
1803        rule->ident = ident;
1804        rule->constraint = constraint;
1805        use_pseudo(insn, pseudo, &rule->pseudo);
1806        add_ptr_list(&insn->asm_rules->outputs, rule);
1807 }

1809 static pseudo_t linearize_asm_statement(struct entrypoint *ep, struct statement
1810 {
1811        int state;
1812        struct expression *expr;
1813        struct instruction *insn;
1814        struct asm_rules *rules;
1815        const char *constraint;
1816        struct ident *ident;

1818        insn = alloc_instruction(OP_ASM, 0);
1819        expr = stmt->asm_string;
1820        if (!expr || expr->type != EXPR_STRING) {
1821                warning(stmt->pos, "expected string in inline asm");
1822                return VOID;
1823        }
1824        insn->string = expr->string->data;

1826        rules = __alloc_asm_rules(0);
1827        insn->asm_rules = rules;

1829        /* Gather the inputs.. */
1830        state = 0;
1831        ident = NULL;
1832        constraint = NULL;
1833        FOR_EACH_PTR(stmt->asm_inputs, expr) {
1834                switch (state) {
1835                case 0: /* Identifier */
1836                        state = 1;
1837                        ident = (struct ident *)expr;
1838                        continue;

1840                case 1: /* Constraint */
1841                        state = 2;
1842                        constraint = expr ? expr->string->data : "";
```

```
1843                            continue;

1845                    case 2: /* Expression */
1846                            state = 0;
1847                            add_asm_input(ep, insn, expr, constraint, ident);
1848                    }
1849            } END_FOR_EACH_PTR(expr);

1851            add_one_insn(ep, insn);

1853            /* Assign the outputs */
1854            state = 0;
1855            ident = NULL;
1856            constraint = NULL;
1857            FOR_EACH_PTR(stmt->asm_outputs, expr) {
1858                    switch (state) {
1859                    case 0: /* Identifier */
1860                            state = 1;
1861                            ident = (struct ident *)expr;
1862                            continue;

1864                    case 1: /* Constraint */
1865                            state = 2;
1866                            constraint = expr ? expr->string->data : "";
1867                            continue;

1869                    case 2:
1870                            state = 0;
1871                            add_asm_output(ep, insn, expr, constraint, ident);
1872                    }
1873            } END_FOR_EACH_PTR(expr);

1875            return VOID;
1876    }

1878    static int multijmp_cmp(const void *_a, const void *_b)
1879    {
1880            const struct multijmp *a = _a;
1881            const struct multijmp *b = _b;

1883            // "default" case?
1884            if (a->begin > a->end) {
1885                    if (b->begin > b->end)
1886                            return 0;
1887                    return 1;
1888            }
1889            if (b->begin > b->end)
1890                    return -1;
1891            if (a->begin == b->begin) {
1892                    if (a->end == b->end)
1893                            return 0;
1894                    return (a->end < b->end) ? -1 : 1;
1895            }
1896            return a->begin < b->begin ? -1 : 1;
1897    }

1899    static void sort_switch_cases(struct instruction *insn)
1900    {
1901            sort_list((struct ptr_list **)&insn->multijmp_list, multijmp_cmp);
1902    }

1904    static pseudo_t linearize_declaration(struct entrypoint *ep, struct statement *s
1905    {
1906            struct symbol *sym;

1908            concat_symbol_list(stmt->declaration, &ep->syms);
```



```
1910            FOR_EACH_PTR(stmt->declaration, sym) {
1911                    linearize_one_symbol(ep, sym);
1912            } END_FOR_EACH_PTR(sym);
1913            return VOID;
1914    }

1916    static pseudo_t linearize_return(struct entrypoint *ep, struct statement *stmt)
1917    {
1918            struct expression *expr = stmt->expression;
1919            struct basic_block *bb_return = get_bound_block(ep, stmt->ret_target);
1920            struct basic_block *active;
1921            pseudo_t src = linearize_expression(ep, expr);
1922            active = ep->active;
1923            if (active && src != VOID) {
1924                    struct instruction *phi_node = first_instruction(bb_return->insn
1925                    pseudo_t phi;
1926                    if (!phi_node) {
1927                            phi_node = alloc_typed_instruction(OP_PHI, expr->ctype);
1928                            phi_node->target = alloc_pseudo(phi_node);
1929                            phi_node->bb = bb_return;
1930                            add_instruction(&bb_return->insns, phi_node);
1931                    }
1932                    phi = alloc_phi(active, src, type_size(expr->ctype));
1933                    phi->ident = &return_ident;
1934                    use_pseudo(phi_node, phi, add_pseudo(&phi_node->phi_list, phi));
1935            }
1936            add_goto(ep, bb_return);
1937            return VOID;
1938    }

1940    static pseudo_t linearize_switch(struct entrypoint *ep, struct statement *stmt)
1941    {
1942            struct symbol *sym;
1943            struct instruction *switch_ins;
1944            struct basic_block *switch_end = alloc_basic_block(ep, stmt->pos);
1945            struct basic_block *active, *default_case;
1946            struct multijmp *jmp;
1947            pseudo_t pseudo;

1949            pseudo = linearize_expression(ep, stmt->switch_expression);

1951            active = ep->active;
1952            if (!bb_reachable(active))
1953                    return VOID;

1955            switch_ins = alloc_instruction(OP_SWITCH, 0);
1956            use_pseudo(switch_ins, pseudo, &switch_ins->cond);
1957            add_one_insn(ep, switch_ins);
1958            finish_block(ep);

1960            default_case = NULL;
1961            FOR_EACH_PTR(stmt->switch_case->symbol_list, sym) {
1962                    struct statement *case_stmt = sym->stmt;
1963                    struct basic_block *bb_case = get_bound_block(ep, sym);

1965                    if (!case_stmt->case_expression) {
1966                            default_case = bb_case;
1967                            continue;
1968                    } else {
1969                            int begin, end;

1971                            begin = end = case_stmt->case_expression->value;
1972                            if (case_stmt->case_to)
1973                                    end = case_stmt->case_to->value;
1974                            if (begin > end)
```

```
1975                                jmp = alloc_multijmp(bb_case, end, begin);
1976                        else
1977                                jmp = alloc_multijmp(bb_case, begin, end);

1979                }
1980                add_multijmp(&switch_ins->multijmp_list, jmp);
1981                add_bb(&bb_case->parents, active);
1982                add_bb(&active->children, bb_case);
1983        } END_FOR_EACH_PTR(sym);

1985        bind_label(stmt->switch_break, switch_end, stmt->pos);

1987        /* And linearize the actual statement */
1988        linearize_statement(ep, stmt->switch_statement);
1989        set_activeblock(ep, switch_end);

1991        if (!default_case)
1992                default_case = switch_end;

1994        jmp = alloc_multijmp(default_case, 1, 0);
1995        add_multijmp(&switch_ins->multijmp_list, jmp);
1996        add_bb(&default_case->parents, active);
1997        add_bb(&active->children, default_case);
1998        sort_switch_cases(switch_ins);

2000        return VOID;
2001 }

2003 static pseudo_t linearize_iterator(struct entrypoint *ep, struct statement *stmt
2004 {
2005        struct statement  *pre_statement = stmt->iterator_pre_statement;
2006        struct expression *pre_condition = stmt->iterator_pre_condition;
2007        struct statement  *statement = stmt->iterator_statement;
2008        struct statement  *post_statement = stmt->iterator_post_statement;
2009        struct expression *post_condition = stmt->iterator_post_condition;
2010        struct basic_block *loop_top, *loop_body, *loop_continue, *loop_end;
2011        struct symbol *sym;

2013        FOR_EACH_PTR(stmt->iterator_syms, sym) {
2014                linearize_one_symbol(ep, sym);
2015        } END_FOR_EACH_PTR(sym);
2016        concat_symbol_list(stmt->iterator_syms, &ep->syms);
2017        linearize_statement(ep, pre_statement);

2019        loop_body = loop_top = alloc_basic_block(ep, stmt->pos);
2020        loop_continue = alloc_basic_block(ep, stmt->pos);
2021        loop_end = alloc_basic_block(ep, stmt->pos);

2023        /* An empty post-condition means that it's the same as the pre-condition
2024        if (!post_condition) {
2025                loop_top = alloc_basic_block(ep, stmt->pos);
2026                set_activeblock(ep, loop_top);
2027        }

2029        if (pre_condition)
2030                        linearize_cond_branch(ep, pre_condition, loop_body, loop

2032        bind_label(stmt->iterator_continue, loop_continue, stmt->pos);
2033        bind_label(stmt->iterator_break, loop_end, stmt->pos);

2035        set_activeblock(ep, loop_body);
2036        linearize_statement(ep, statement);
2037        add_goto(ep, loop_continue);

2039        set_activeblock(ep, loop_continue);
2040        linearize_statement(ep, post_statement);
```



```
2041        if (!post_condition)
2042                add_goto(ep, loop_top);
2043        else
2044                linearize_cond_branch(ep, post_condition, loop_top, loop_end);
2045        set_activeblock(ep, loop_end);

2047        return VOID;
2048 }

2050 pseudo_t linearize_statement(struct entrypoint *ep, struct statement *stmt)
2051 {
2052        struct basic_block *bb;

2054        if (!stmt)
2055                return VOID;

2057        bb = ep->active;
2058        if (bb && !bb->insns)
2059                bb->pos = stmt->pos;
2060        current_pos = stmt->pos;

2062        switch (stmt->type) {
2063        case STMT_NONE:
2064                break;

2066        case STMT_DECLARATION:
2067                return linearize_declaration(ep, stmt);

2069        case STMT_CONTEXT:
2070                return linearize_context(ep, stmt);

2072        case STMT_RANGE:
2073                return linearize_range(ep, stmt);

2075        case STMT_EXPRESSION:
2076                return linearize_expression(ep, stmt->expression);

2078        case STMT_ASM:
2079                return linearize_asm_statement(ep, stmt);

2081        case STMT_RETURN:
2082                return linearize_return(ep, stmt);

2084        case STMT_CASE: {
2085                add_label(ep, stmt->case_label);
2086                linearize_statement(ep, stmt->case_statement);
2087                break;
2088        }

2090        case STMT_LABEL: {
2091                struct symbol *label = stmt->label_identifier;

2093                if (label->used) {
2094                        add_label(ep, label);
2095                }
2096                return linearize_statement(ep, stmt->label_statement);
2097        }

2099        case STMT_GOTO: {
2100                struct symbol *sym;
2101                struct expression *expr;
2102                struct instruction *goto_ins;
2103                struct basic_block *active;
2104                pseudo_t pseudo;

2106                active = ep->active;
```

```
2107                if (!bb_reachable(active))
2108                        break;

2110                if (stmt->goto_label) {
2111                        add_goto(ep, get_bound_block(ep, stmt->goto_label));
2112                        break;
2113                }

2115                expr = stmt->goto_expression;
2116                if (!expr)
2117                        break;

2119                /* This can happen as part of simplification */
2120                if (expr->type == EXPR_LABEL) {
2121                        add_goto(ep, get_bound_block(ep, expr->label_symbol));
2122                        break;
2123                }

2125                pseudo = linearize_expression(ep, expr);
2126                goto_ins = alloc_instruction(OP_COMPUTEDGOTO, 0);
2127                use_pseudo(goto_ins, pseudo, &goto_ins->target);
2128                add_one_insn(ep, goto_ins);

2130                FOR_EACH_PTR(stmt->target_list, sym) {
2131                        struct basic_block *bb_computed = get_bound_block(ep, sy
2132                        struct multijmp *jmp = alloc_multijmp(bb_computed, 1, 0)
2133                        add_multijmp(&goto_ins->multijmp_list, jmp);
2134                        add_bb(&bb_computed->parents, ep->active);
2135                        add_bb(&active->children, bb_computed);
2136                } END_FOR_EACH_PTR(sym);

2138                finish_block(ep);
2139                break;
2140        }

2142        case STMT_COMPOUND:
2143                if (stmt->inline_fn)
2144                        return linearize_inlined_call(ep, stmt);
2145                return linearize_compound_statement(ep, stmt);

2147        /*
2148         * This could take 'likely/unlikely' into account, and
2149         * switch the arms around appropriately..
2150         */
2151        case STMT_IF: {
2152                struct basic_block *bb_true, *bb_false, *endif;
2153                struct expression *cond = stmt->if_conditional;

2155                bb_true = alloc_basic_block(ep, stmt->pos);
2156                bb_false = endif = alloc_basic_block(ep, stmt->pos);

2158                linearize_cond_branch(ep, cond, bb_true, bb_false);

2160                set_activeblock(ep, bb_true);
2161                linearize_statement(ep, stmt->if_true);
2162
2163                if (stmt->if_false) {
2164                        endif = alloc_basic_block(ep, stmt->pos);
2165                        add_goto(ep, endif);
2166                        set_activeblock(ep, bb_false);
2167                        linearize_statement(ep, stmt->if_false);
2168                }
2169                set_activeblock(ep, endif);
2170                break;
2171        }
```



```
2173        case STMT_SWITCH:
2174                return linearize_switch(ep, stmt);

2176        case STMT_ITERATOR:
2177                return linearize_iterator(ep, stmt);

2179        default:
2180                break;
2181        }
2182        return VOID;
2183 }

2185 static struct entrypoint *linearize_fn(struct symbol *sym, struct symbol *base_t
2186 {
2187        struct entrypoint *ep;
2188        struct basic_block *bb;
2189        struct symbol *arg;
2190        struct instruction *entry;
2191        pseudo_t result;
2192        int i;

2194        if (!base_type->stmt)
2195                return NULL;

2197        ep = alloc_entrypoint();
2198        bb = alloc_basic_block(ep, sym->pos);
2199
2200        ep->name = sym;
2201        sym->ep = ep;
2202        set_activeblock(ep, bb);

2204        entry = alloc_instruction(OP_ENTRY, 0);
2205        add_one_insn(ep, entry);
2206        ep->entry = entry;

2208        concat_symbol_list(base_type->arguments, &ep->syms);

2210        /* FIXME!! We should do something else about varargs.. */
2211        i = 0;
2212        FOR_EACH_PTR(base_type->arguments, arg) {
2213                linearize_argument(ep, arg, ++i);
2214        } END_FOR_EACH_PTR(arg);

2216        result = linearize_statement(ep, base_type->stmt);
2217        if (bb_reachable(ep->active) && !bb_terminated(ep->active)) {
2218                struct symbol *ret_type = base_type->ctype.base_type;
2219                struct instruction *insn = alloc_typed_instruction(OP_RET, ret_t

2221                if (type_size(ret_type) > 0)
2222                        use_pseudo(insn, result, &insn->src);
2223                add_one_insn(ep, insn);
2224        }

2226        if (fdump_linearize) {
2227                if (fdump_linearize == 2)
2228                        return ep;
2229                show_entry(ep);
2230        }

2232        /*
2233         * Do trivial flow simplification - branches to
2234         * branches, kill dead basicblocks etc
2235         */
2236        kill_unreachable_bbs(ep);

2238        /*
```

```
2239          * Turn symbols into pseudos
2240          */
2241         simplify_symbol_usage(ep);

2243 repeat:
2244         /*
2245          * Remove trivial instructions, and try to CSE
2246          * the rest.
2247          */
2248         do {
2249                 cleanup_and_cse(ep);
2250                 pack_basic_blocks(ep);
2251         } while (repeat_phase & REPEAT_CSE);

2253         kill_unreachable_bbs(ep);
2254         vrfy_flow(ep);

2256         /* Cleanup */
2257         clear_symbol_pseudos(ep);

2259         /* And track pseudo register usage */
2260         track_pseudo_liveness(ep);

2262         /*
2263          * Some flow optimizations can only effectively
2264          * be done when we've done liveness analysis. But
2265          * if they trigger, we need to start all over
2266          * again
2267          */
2268         if (simplify_flow(ep)) {
2269                 clear_liveness(ep);
2270                 goto repeat;
2271         }

2273         /* Finally, add deathnotes to pseudos now that we have them */
2274         if (dbg_dead)
2275                 track_pseudo_death(ep);

2277         return ep;
2278 }

2280 struct entrypoint *linearize_symbol(struct symbol *sym)
2281 {
2282         struct symbol *base_type;

2284         if (!sym)
2285                 return NULL;
2286         current_pos = sym->pos;
2287         base_type = sym->ctype.base_type;
2288         if (!base_type)
2289                 return NULL;
2290         if (base_type->type == SYM_FN)
2291                 return linearize_fn(sym, base_type);
2292         return NULL;
2293 }
```

```
**********************************************************
    7113 Fri Dec 21 15:00:13 2018
new/usr/src/tools/smatch/src/linearize.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef LINEARIZE_H
#define LINEARIZE_H

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"

struct instruction;
DECLARE_PTR_LIST(pseudo_ptr_list, pseudo_t);

struct pseudo_user {
        struct instruction *insn;
        pseudo_t *userp;
};

DECLARE_ALLOCATOR(pseudo_user);
DECLARE_PTR_LIST(pseudo_user_list, struct pseudo_user);


enum pseudo_type {
        PSEUDO_VOID,
        PSEUDO_REG,
        PSEUDO_SYM,
        PSEUDO_VAL,
        PSEUDO_ARG,
        PSEUDO_PHI,
};

struct pseudo {
        int nr;
        int size:16;    /* OP_SETVAL only */
        enum pseudo_type type:8;
        struct pseudo_user_list *users;
        struct ident *ident;
        union {
                struct symbol *sym;
                struct instruction *def;
                long long value;
        };
        void *priv;
};

extern struct pseudo void_pseudo;

#define VOID (&void_pseudo)

struct multijmp {
        struct basic_block *target;
        int begin, end;
};

struct asm_constraint {
        pseudo_t pseudo;
        const char *constraint;
        const struct ident *ident;
};

DECLARE_ALLOCATOR(asm_constraint);
```



```
DECLARE_PTR_LIST(asm_constraint_list, struct asm_constraint);

struct asm_rules {
        struct asm_constraint_list *inputs;
        struct asm_constraint_list *outputs;
        struct asm_constraint_list *clobbers;
};

DECLARE_ALLOCATOR(asm_rules);

struct instruction {
        unsigned opcode:8,
                 size:24;
        struct basic_block *bb;
        struct position pos;
        struct symbol *type;
        union {
                pseudo_t target;
                pseudo_t cond;          /* for branch and switch */
        };
        union {
                struct /* entrypoint */ {
                        struct pseudo_list *arg_list;
                };
                struct /* branch */ {
                        struct basic_block *bb_true, *bb_false;
                };
                struct /* switch */ {
                        struct multijmp_list *multijmp_list;
                };
                struct /* phi_node */ {
                        struct pseudo_list *phi_list;
                };
                struct /* phi source */ {
                        pseudo_t phi_src;
                        struct instruction_list *phi_users;
                };
                struct /* unops */ {
                        pseudo_t src;
                        struct symbol *orig_type;       /* casts */
                        unsigned int offset;            /* memops */
                };
                struct /* binops and sel */ {
                        pseudo_t src1, src2, src3;
                };
                struct /* slice */ {
                        pseudo_t base;
                        unsigned from, len;
                };
                struct /* setval */ {
                        pseudo_t symbol;                /* Subtle: same offset a
                        struct expression *val;
                };
                struct /* call */ {
                        pseudo_t func;
                        struct pseudo_list *arguments;
                        struct symbol *fntype;
                };
                struct /* context */ {
                        int increment;
                        int check;
                        struct expression *context_expr;
                };
                struct /* asm */ {
                        const char *string;
                        struct asm_rules *asm_rules;
```

```
127                 };
128         };
129 };

131 enum opcode {
132         OP_BADOP,

134         /* Entry */
135         OP_ENTRY,

137         /* Terminator */
138         OP_TERMINATOR,
139         OP_RET = OP_TERMINATOR,
140         OP_BR,
141         OP_CBR,
142         OP_SWITCH,
143         OP_INVOKE,
144         OP_COMPUTEDGOTO,
145         OP_UNWIND,
146         OP_TERMINATOR_END = OP_UNWIND,
147
148         /* Binary */
149         OP_BINARY,
150         OP_ADD = OP_BINARY,
151         OP_SUB,
152         OP_MULU, OP_MULS,
153         OP_DIVU, OP_DIVS,
154         OP_MODU, OP_MODS,
155         OP_SHL,
156         OP_LSR, OP_ASR,
157
158         /* Logical */
159         OP_AND,
160         OP_OR,
161         OP_XOR,
162         OP_AND_BOOL,
163         OP_OR_BOOL,
164         OP_BINARY_END = OP_OR_BOOL,

166         /* Binary comparison */
167         OP_BINCMP,
168         OP_SET_EQ = OP_BINCMP,
169         OP_SET_NE,
170         OP_SET_LE,
171         OP_SET_GE,
172         OP_SET_LT,
173         OP_SET_GT,
174         OP_SET_B,
175         OP_SET_A,
176         OP_SET_BE,
177         OP_SET_AE,
178         OP_BINCMP_END = OP_SET_AE,

180         /* Uni */
181         OP_NOT,
182         OP_NEG,

184         /* Select - three input values */
185         OP_SEL,
186
187         /* Memory */
188         OP_MALLOC,
189         OP_FREE,
190         OP_ALLOCA,
191         OP_LOAD,
192         OP_STORE,
```



```
193         OP_SETVAL,
194         OP_SYMADDR,
195         OP_GET_ELEMENT_PTR,

197         /* Other */
198         OP_PHI,
199         OP_PHISOURCE,
200         OP_CAST,
201         OP_SCAST,
202         OP_FPCAST,
203         OP_PTRCAST,
204         OP_INLINED_CALL,
205         OP_CALL,
206         OP_VANEXT,
207         OP_VAARG,
208         OP_SLICE,
209         OP_SNOP,
210         OP_LNOP,
211         OP_NOP,
212         OP_DEATHNOTE,
213         OP_ASM,

215         /* Sparse tagging (line numbers, context, whatever) */
216         OP_CONTEXT,
217         OP_RANGE,

219         /* Needed to translate SSA back to normal form */
220         OP_COPY,
221 };

223 struct basic_block_list;
224 struct instruction_list;

226 struct basic_block {
227         struct position pos;
228         unsigned long generation;
229         int context;
230         struct entrypoint *ep;
231         struct basic_block_list *parents; /* sources */
232         struct basic_block_list *children; /* destinations */
233         struct instruction_list *insns; /* Linear list of instructions */
234         struct pseudo_list *needs, *defines;
235         union {
236                 unsigned int nr;        /* unique id for label's names */
237                 void *priv;
238         };
239 };


242 static inline void add_bb(struct basic_block_list **list, struct basic_block *bb
243 {
244         add_ptr_list(list, bb);
245 }

247 static inline void add_instruction(struct instruction_list **list, struct instru
248 {
249         add_ptr_list(list, insn);
250 }

252 static inline void add_multijmp(struct multijmp_list **list, struct multijmp *mu
253 {
254         add_ptr_list(list, multijmp);
255 }

257 static inline pseudo_t *add_pseudo(struct pseudo_list **list, pseudo_t pseudo)
258 {
```

```
259         return add_ptr_list(list, pseudo);
260 }

262 static inline int remove_pseudo(struct pseudo_list **list, pseudo_t pseudo)
263 {
264         return delete_ptr_list_entry((struct ptr_list **)list, pseudo, 0) != 0;
265 }

267 static inline int bb_terminated(struct basic_block *bb)
268 {
269         struct instruction *insn;
270         if (!bb)
271                 return 0;
272         insn = last_instruction(bb->insns);
273         return insn && insn->opcode >= OP_TERMINATOR
274                     && insn->opcode <= OP_TERMINATOR_END;
275 }

277 static inline int bb_reachable(struct basic_block *bb)
278 {
279         return bb != NULL;
280 }

282 static inline void add_pseudo_ptr(pseudo_t *ptr, struct pseudo_ptr_list **list)
283 {
284         add_ptr_list(list, ptr);
285 }

287 static inline void add_pseudo_user_ptr(struct pseudo_user *user, struct pseudo_u
288 {
289         add_ptr_list(list, user);
290 }

292 static inline int has_use_list(pseudo_t p)
293 {
294         return (p && p->type != PSEUDO_VOID && p->type != PSEUDO_VAL);
295 }

297 static inline struct pseudo_user *alloc_pseudo_user(struct instruction *insn, ps
298 {
299         struct pseudo_user *user = __alloc_pseudo_user(0);
300         user->userp = pp;
301         user->insn = insn;
302         return user;
303 }

305 static inline void use_pseudo(struct instruction *insn, pseudo_t p, pseudo_t *pp
306 {
307         *pp = p;
308         if (has_use_list(p))
309                 add_pseudo_user_ptr(alloc_pseudo_user(insn, pp), &p->users);
310 }

312 static inline void remove_bb_from_list(struct basic_block_list **list, struct ba
313 {
314         delete_ptr_list_entry((struct ptr_list **)list, entry, count);
315 }

317 static inline void replace_bb_in_list(struct basic_block_list **list,
318         struct basic_block *old, struct basic_block *new, int count)
319 {
320         replace_ptr_list_entry((struct ptr_list **)list, old, new, count);
321 }

323 struct entrypoint {
324         struct symbol *name;
```



```
325         struct symbol_list *syms;
326         struct pseudo_list *accesses;
327         struct basic_block_list *bbs;
328         struct basic_block *active;
329         struct instruction *entry;
330 };

332 extern void insert_select(struct basic_block *bb, struct instruction *br, struct
333 extern void insert_branch(struct basic_block *bb, struct instruction *br, struct

335 pseudo_t alloc_phi(struct basic_block *source, pseudo_t pseudo, int size);
336 pseudo_t alloc_pseudo(struct instruction *def);
337 pseudo_t value_pseudo(struct symbol *type, long long val);
338 unsigned int value_size(long long value);

340 struct entrypoint *linearize_symbol(struct symbol *sym);
341 int unssa(struct entrypoint *ep);
342 void show_entry(struct entrypoint *ep);
343 const char *show_pseudo(pseudo_t pseudo);
344 void show_bb(struct basic_block *bb);
345 const char *show_instruction(struct instruction *insn);

347 #endif /* LINEARIZE_H */
```

```
**********************************************************
    7731 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/liveness.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Register - track pseudo usage, maybe eventually try to do register
 * allocation.
 *
 * Copyright (C) 2004 Linus Torvalds
 */

#include <assert.h>

#include "parse.h"
#include "expression.h"
#include "linearize.h"
#include "flow.h"

static void phi_defines(struct instruction * phi_node, pseudo_t target,
        void (*defines)(struct basic_block *, pseudo_t))
{
        pseudo_t phi;
        FOR_EACH_PTR(phi_node->phi_list, phi) {
                struct instruction *def;
                if (phi == VOID)
                        continue;
                def = phi->def;
                if (!def || !def->bb)
                        continue;
                defines(def->bb, target);
        } END_FOR_EACH_PTR(phi);
}

static void asm_liveness(struct basic_block *bb, struct instruction *insn,
        void (*def)(struct basic_block *, pseudo_t),
        void (*use)(struct basic_block *, pseudo_t))
{
        struct asm_constraint *entry;

        FOR_EACH_PTR(insn->asm_rules->inputs, entry) {
                use(bb, entry->pseudo);
        } END_FOR_EACH_PTR(entry);

        FOR_EACH_PTR(insn->asm_rules->outputs, entry) {
                def(bb, entry->pseudo);
        } END_FOR_EACH_PTR(entry);
}

static void track_instruction_usage(struct basic_block *bb, struct instruction *
        void (*def)(struct basic_block *, pseudo_t),
        void (*use)(struct basic_block *, pseudo_t))
{
        pseudo_t pseudo;

        #define USES(x)         use(bb, insn->x)
        #define DEFINES(x)      def(bb, insn->x)

        switch (insn->opcode) {
        case OP_RET:
                USES(src);
                break;

        case OP_CBR:
        case OP_SWITCH:
```



```
                USES(cond);
                break;

        case OP_COMPUTEDGOTO:
                USES(target);
                break;

        /* Binary */
        case OP_BINARY ... OP_BINARY_END:
        case OP_BINCMP ... OP_BINCMP_END:
                USES(src1); USES(src2); DEFINES(target);
                break;

        /* Uni */
        case OP_NOT: case OP_NEG:
                USES(src1); DEFINES(target);
                break;

        case OP_SEL:
                USES(src1); USES(src2); USES(src3); DEFINES(target);
                break;

        /* Memory */
        case OP_LOAD:
                USES(src); DEFINES(target);
                break;

        case OP_STORE:
                USES(src); USES(target);
                break;

        case OP_SETVAL:
                DEFINES(target);
                break;

        case OP_SYMADDR:
                USES(symbol); DEFINES(target);
                break;

        /* Other */
        case OP_PHI:
                /* Phi-nodes are "backwards" nodes. Their def doesn't matter */
                phi_defines(insn, insn->target, def);
                break;

        case OP_PHISOURCE:
                /*
                 * We don't care about the phi-source define, they get set
                 * up and expanded by the OP_PHI
                 */
                USES(phi_src);
                break;

        case OP_CAST:
        case OP_SCAST:
        case OP_FPCAST:
        case OP_PTRCAST:
                USES(src); DEFINES(target);
                break;

        case OP_CALL:
                USES(func);
                if (insn->target != VOID)
                        DEFINES(target);
                FOR_EACH_PTR(insn->arguments, pseudo) {
                        use(bb, pseudo);
```

```
127                 } END_FOR_EACH_PTR(pseudo);
128                 break;

130         case OP_SLICE:
131                 USES(base); DEFINES(target);
132                 break;

134         case OP_ASM:
135                 asm_liveness(bb, insn, def, use);
136                 break;

138         case OP_RANGE:
139                 USES(src1); USES(src2); USES(src3);
140                 break;

142         case OP_BADOP:
143         case OP_INVOKE:
144         case OP_UNWIND:
145         case OP_MALLOC:
146         case OP_FREE:
147         case OP_ALLOCA:
148         case OP_GET_ELEMENT_PTR:
149         case OP_VANEXT:
150         case OP_VAARG:
151         case OP_SNOP:
152         case OP_LNOP:
153         case OP_NOP:
154         case OP_CONTEXT:
155                 break;
156         }
157 }

159 int pseudo_in_list(struct pseudo_list *list, pseudo_t pseudo)
160 {
161         pseudo_t old;
162         FOR_EACH_PTR(list,old) {
163                 if (old == pseudo)
164                         return 1;
165         } END_FOR_EACH_PTR(old);
166         return 0;
167 }

169 static int liveness_changed;

171 static void add_pseudo_exclusive(struct pseudo_list **list, pseudo_t pseudo)
172 {
173         if (!pseudo_in_list(*list, pseudo)) {
174                 liveness_changed = 1;
175                 add_pseudo(list, pseudo);
176         }
177 }

179 static inline int trackable_pseudo(pseudo_t pseudo)
180 {
181         return pseudo && (pseudo->type == PSEUDO_REG || pseudo->type == PSEUDO_A
182 }

184 static void insn_uses(struct basic_block *bb, pseudo_t pseudo)
185 {
186         if (trackable_pseudo(pseudo)) {
187                 struct instruction *def = pseudo->def;
188                 if (pseudo->type != PSEUDO_REG || def->bb != bb || def->opcode =
189                         add_pseudo_exclusive(&bb->needs, pseudo);
190         }
191 }
```



```
193 static void insn_defines(struct basic_block *bb, pseudo_t pseudo)
194 {
195         assert(trackable_pseudo(pseudo));
196         add_pseudo(&bb->defines, pseudo);
197 }

199 static void track_bb_liveness(struct basic_block *bb)
200 {
201         pseudo_t needs;

203         FOR_EACH_PTR(bb->needs, needs) {
204                 struct basic_block *parent;
205                 FOR_EACH_PTR(bb->parents, parent) {
206                         if (!pseudo_in_list(parent->defines, needs)) {
207                                 add_pseudo_exclusive(&parent->needs, needs);
208                         }
209                 } END_FOR_EACH_PTR(parent);
210         } END_FOR_EACH_PTR(needs);
211 }

213 /*
214  * We need to clear the liveness information if we
215  * are going to re-run it.
216  */
217 void clear_liveness(struct entrypoint *ep)
218 {
219         struct basic_block *bb;

221         FOR_EACH_PTR(ep->bbs, bb) {
222                 free_ptr_list(&bb->needs);
223                 free_ptr_list(&bb->defines);
224         } END_FOR_EACH_PTR(bb);
225 }

227 /*
228  * Track inter-bb pseudo liveness. The intra-bb case
229  * is purely local information.
230  */
231 void track_pseudo_liveness(struct entrypoint *ep)
232 {
233         struct basic_block *bb;

235         /* Add all the bb pseudo usage */
236         FOR_EACH_PTR(ep->bbs, bb) {
237                 struct instruction *insn;
238                 FOR_EACH_PTR(bb->insns, insn) {
239                         if (!insn->bb)
240                                 continue;
241                         assert(insn->bb == bb);
242                         track_instruction_usage(bb, insn, insn_defines, insn_use
243                 } END_FOR_EACH_PTR(insn);
244         } END_FOR_EACH_PTR(bb);

246         /* Calculate liveness.. */
247         do {
248                 liveness_changed = 0;
249                 FOR_EACH_PTR_REVERSE(ep->bbs, bb) {
250                         track_bb_liveness(bb);
251                 } END_FOR_EACH_PTR_REVERSE(bb);
252         } while (liveness_changed);

254         /* Remove the pseudos from the "defines" list that are used internally *
255         FOR_EACH_PTR(ep->bbs, bb) {
256                 pseudo_t def;
257                 FOR_EACH_PTR(bb->defines, def) {
258                         struct basic_block *child;
```

```
259                         FOR_EACH_PTR(bb->children, child) {
260                                 if (pseudo_in_list(child->needs, def))
261                                         goto is_used;
262                         } END_FOR_EACH_PTR(child);
263                         DELETE_CURRENT_PTR(def);
264 is_used:
265                 ;
266                 } END_FOR_EACH_PTR(def);
267                 PACK_PTR_LIST(&bb->defines);
268         } END_FOR_EACH_PTR(bb);
269 }

271 static void merge_pseudo_list(struct pseudo_list *src, struct pseudo_list **dest
272 {
273         pseudo_t pseudo;
274         FOR_EACH_PTR(src, pseudo) {
275                 add_pseudo_exclusive(dest, pseudo);
276         } END_FOR_EACH_PTR(pseudo);
277 }

279 void track_phi_uses(struct instruction *insn)
280 {
281         pseudo_t phi;
282         FOR_EACH_PTR(insn->phi_list, phi) {
283                 struct instruction *def;
284                 if (phi == VOID || !phi->def)
285                         continue;
286                 def = phi->def;
287                 assert(def->opcode == OP_PHISOURCE);
288                 add_ptr_list(&def->phi_users, insn);
289         } END_FOR_EACH_PTR(phi);
290 }

292 static void track_bb_phi_uses(struct basic_block *bb)
293 {
294         struct instruction *insn;
295         FOR_EACH_PTR(bb->insns, insn) {
296                 if (insn->bb && insn->opcode == OP_PHI)
297                         track_phi_uses(insn);
298         } END_FOR_EACH_PTR(insn);
299 }

301 static struct pseudo_list **live_list;
302 static struct pseudo_list *dead_list;

304 static void death_def(struct basic_block *bb, pseudo_t pseudo)
305 {
306 }

308 static void death_use(struct basic_block *bb, pseudo_t pseudo)
309 {
310         if (trackable_pseudo(pseudo) && !pseudo_in_list(*live_list, pseudo)) {
311                 add_pseudo(&dead_list, pseudo);
312                 add_pseudo(live_list, pseudo);
313         }
314 }

316 static void track_pseudo_death_bb(struct basic_block *bb)
317 {
318         struct pseudo_list *live = NULL;
319         struct basic_block *child;
320         struct instruction *insn;

322         FOR_EACH_PTR(bb->children, child) {
323                 merge_pseudo_list(child->needs, &live);
324         } END_FOR_EACH_PTR(child);
```



```
326         live_list = &live;
327         FOR_EACH_PTR_REVERSE(bb->insns, insn) {
328                 if (!insn->bb)
329                         continue;

331                 dead_list = NULL;
332                 track_instruction_usage(bb, insn, death_def, death_use);
333                 if (dead_list) {
334                         pseudo_t dead;
335                         FOR_EACH_PTR(dead_list, dead) {
336                                 struct instruction *deathnote = __alloc_instruct
337                                 deathnote->bb = bb;
338                                 deathnote->opcode = OP_DEATHNOTE;
339                                 deathnote->target = dead;
340                                 INSERT_CURRENT(deathnote, insn);
341                         } END_FOR_EACH_PTR(dead);
342                         free_ptr_list(&dead_list);
343                 }
344         } END_FOR_EACH_PTR_REVERSE(insn);
345         free_ptr_list(&live);
346 }

348 void track_pseudo_death(struct entrypoint *ep)
349 {
350         struct basic_block *bb;

352         FOR_EACH_PTR(ep->bbs, bb) {
353                 track_bb_phi_uses(bb);
354         } END_FOR_EACH_PTR(bb);

356         FOR_EACH_PTR(ep->bbs, bb) {
357                 track_pseudo_death_bb(bb);
358         } END_FOR_EACH_PTR(bb);
359 }
```

```
**********************************************************
    2157 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/macro_table.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/macro_table.c
 *
 * Copyright (C) 2010 Dan Carpenter.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "lib.h"
#include "parse.h"
#include "cwchash/hashtable.h"

static struct hashtable *macro_table;

static DEFINE_HASHTABLE_INSERT(do_insert_macro, struct position, char);
static DEFINE_HASHTABLE_SEARCH(do_search_macro, struct position, char);

static inline unsigned int position_hash(void *_pos)
{
        struct position *pos = _pos;

        return pos->line | (pos->pos << 22) | (pos->stream << 18);
}

static inline int equalkeys(void *_pos1, void *_pos2)
{
        struct position *pos1 = _pos1;
        struct position *pos2 = _pos2;

        return pos1->line == pos2->line && pos1->pos == pos2->pos &&
                pos1->stream == pos2->stream;
}

void store_macro_pos(struct token *token)
{
        if (!macro_table)
                macro_table = create_hashtable(5000, position_hash, equalkeys);

        if (get_macro_name(token->pos))
                return;
```



```
        do_insert_macro(macro_table, &token->pos, token->ident->name);
}

char *get_macro_name(struct position pos)
{
        return do_search_macro(macro_table, &pos);
}
```

```
**********************************************************
    4675 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/memops.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * memops - try to combine memory ops.
 *
 * Copyright (C) 2004 Linus Torvalds
 */

#include <string.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <stddef.h>
#include <assert.h>

#include "parse.h"
#include "expression.h"
#include "linearize.h"
#include "flow.h"

static int find_dominating_parents(pseudo_t pseudo, struct instruction *insn,
        struct basic_block *bb, unsigned long generation, struct pseudo_list **d
        int local)
{
        struct basic_block *parent;

        FOR_EACH_PTR(bb->parents, parent) {
                struct instruction *one;
                struct instruction *br;
                pseudo_t phi;

                FOR_EACH_PTR_REVERSE(parent->insns, one) {
                        int dominance;
                        if (!one->bb)
                                continue;
                        if (one == insn)
                                goto no_dominance;
                        dominance = dominates(pseudo, insn, one, local);
                        if (dominance < 0) {
                                if (one->opcode == OP_LOAD)
                                        continue;
                                return 0;
                        }
                        if (!dominance)
                                continue;
                        goto found_dominator;
                } END_FOR_EACH_PTR_REVERSE(one);
no_dominance:
                if (parent->generation == generation)
                        continue;
                parent->generation = generation;

                if (!find_dominating_parents(pseudo, insn, parent, generation, d
                        return 0;
                continue;

found_dominator:
                br = delete_last_instruction(&parent->insns);
                phi = alloc_phi(parent, one->target, one->size);
                phi->ident = phi->ident ? : one->target->ident;
                add_instruction(&parent->insns, br);
                use_pseudo(insn, phi, add_pseudo(dominators, phi));
```



```
        } END_FOR_EACH_PTR(parent);
        return 1;
}

static int address_taken(pseudo_t pseudo)
{
        struct pseudo_user *pu;
        FOR_EACH_PTR(pseudo->users, pu) {
                struct instruction *insn = pu->insn;
                if (insn->bb && (insn->opcode != OP_LOAD && insn->opcode != OP_S
                        return 1;
        } END_FOR_EACH_PTR(pu);
        return 0;
}

static int local_pseudo(pseudo_t pseudo)
{
        return pseudo->type == PSEUDO_SYM
                && !(pseudo->sym->ctype.modifiers & (MOD_STATIC | MOD_NONLOCAL))
                && !address_taken(pseudo);
}

static void simplify_loads(struct basic_block *bb)
{
        struct instruction *insn;

        FOR_EACH_PTR_REVERSE(bb->insns, insn) {
                if (!insn->bb)
                        continue;
                if (insn->opcode == OP_LOAD) {
                        struct instruction *dom;
                        pseudo_t pseudo = insn->src;
                        int local = local_pseudo(pseudo);
                        struct pseudo_list *dominators;
                        unsigned long generation;

                        /* Check for illegal offsets.. */
                        check_access(insn);

                        if (insn->type->ctype.modifiers & MOD_VOLATILE)
                                continue;

                        RECURSE_PTR_REVERSE(insn, dom) {
                                int dominance;
                                if (!dom->bb)
                                        continue;
                                dominance = dominates(pseudo, insn, dom, local);
                                if (dominance) {
                                        /* possible partial dominance? */
                                        if (dominance < 0)  {
                                                if (dom->opcode == OP_LOAD)
                                                        continue;
                                                goto next_load;
                                        }
                                        /* Yeehaa! Found one! */
                                        convert_load_instruction(insn, dom->targ
                                        goto next_load;
                                }
                        } END_FOR_EACH_PTR_REVERSE(dom);

                        /* OK, go find the parents */
                        generation = ++bb_generation;
                        bb->generation = generation;
                        dominators = NULL;
                        if (find_dominating_parents(pseudo, insn, bb, generation
                                /* This happens with initial assignments to stru
```

```
127                                 if (!dominators) {
128                                         if (local) {
129                                                 assert(pseudo->type != PSEUDO_AR
130                                                 convert_load_instruction(insn, v
131                                         }
132                                         goto next_load;
133                                 }
134                                 rewrite_load_instruction(insn, dominators);
135                         }
136                 }
137 next_load:
138                 /* Do the next one */;
139         } END_FOR_EACH_PTR_REVERSE(insn);
140 }

142 static void kill_store(struct instruction *insn)
143 {
144         if (insn) {
145                 insn->bb = NULL;
146                 insn->opcode = OP_SNOP;
147                 kill_use(&insn->target);
148         }
149 }

151 static void kill_dominated_stores(struct basic_block *bb)
152 {
153         struct instruction *insn;

155         FOR_EACH_PTR_REVERSE(bb->insns, insn) {
156                 if (!insn->bb)
157                         continue;
158                 if (insn->opcode == OP_STORE) {
159                         struct instruction *dom;
160                         pseudo_t pseudo = insn->src;
161                         int local = local_pseudo(pseudo);

163                         RECURSE_PTR_REVERSE(insn, dom) {
164                                 int dominance;
165                                 if (!dom->bb)
166                                         continue;
167                                 dominance = dominates(pseudo, insn, dom, local);
168                                 if (dominance) {
169                                         /* possible partial dominance? */
170                                         if (dominance < 0)
171                                                 goto next_store;
172                                         if (dom->opcode == OP_LOAD)
173                                                 goto next_store;
174                                         /* Yeehaa! Found one! */
175                                         kill_store(dom);
176                                 }
177                         } END_FOR_EACH_PTR_REVERSE(dom);

179                         /* OK, we should check the parents now */
180                 }
181 next_store:
182                 /* Do the next one */;
183         } END_FOR_EACH_PTR_REVERSE(insn);
184 }

186 void simplify_memops(struct entrypoint *ep)
187 {
188         struct basic_block *bb;

190         FOR_EACH_PTR_REVERSE(ep->bbs, bb) {
191                 simplify_loads(bb);
192         } END_FOR_EACH_PTR_REVERSE(bb);
```



```
194         FOR_EACH_PTR_REVERSE(ep->bbs, bb) {
195                 kill_dominated_stores(bb);
196         } END_FOR_EACH_PTR_REVERSE(bb);
197 }
```

```
**********************************************************
    2223 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/obfuscate.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Example trivial client program that uses the sparse library
 * to tokenize, preprocess and parse a C file, and prints out
 * the results.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"
#include "linearize.h"

static void emit_entrypoint(struct entrypoint *ep)
{

}

static void emit_symbol(struct symbol *sym)
{
        struct entrypoint *ep;
        ep = linearize_symbol(sym);
        if (ep)
                emit_entrypoint(ep);
}

static void emit_symbol_list(struct symbol_list *list)
{
        struct symbol *sym;

        FOR_EACH_PTR(list, sym) {
```



```
                expand_symbol(sym);
                emit_symbol(sym);
        } END_FOR_EACH_PTR(sym);
}

int main(int argc, char **argv)
{
        struct string_list *filelist = NULL;
        char *file;

        emit_symbol_list(sparse_initialize(argc, argv, &filelist));
        FOR_EACH_PTR_NOTAG(filelist, file) {
                emit_symbol_list(sparse(file));
        } END_FOR_EACH_PTR_NOTAG(file);
        return 0;
}
```

```
**********************************************************
   83310 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/parse.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Stupid C parser, version 1e-6.
 *
 * Let's see how hard this is to do.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 * Copyright (C) 2004 Christopher Li
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <limits.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "scope.h"
#include "expression.h"
#include "target.h"

static struct symbol_list **function_symbol_list;
struct symbol_list *function_computed_target_list;
struct statement_list *function_computed_goto_list;

static struct token *statement(struct token *token, struct statement **tree);
static struct token *handle_attributes(struct token *token, struct decl_state *c

typedef struct token *declarator_t(struct token *, struct decl_state *);
static declarator_t
        struct_specifier, union_specifier, enum_specifier,
        attribute_specifier, typeof_specifier, parse_asm_declarator,
        typedef_specifier, inline_specifier, auto_specifier,
        register_specifier, static_specifier, extern_specifier,
        thread_specifier, const_qualifier, volatile_qualifier;
```



```
static struct token *parse_if_statement(struct token *token, struct statement *s
static struct token *parse_return_statement(struct token *token, struct statemen
static struct token *parse_loop_iterator(struct token *token, struct statement *
static struct token *parse_default_statement(struct token *token, struct stateme
static struct token *parse_case_statement(struct token *token, struct statement
static struct token *parse_switch_statement(struct token *token, struct statemen
static struct token *parse_for_statement(struct token *token, struct statement *
static struct token *parse_while_statement(struct token *token, struct statement
static struct token *parse_do_statement(struct token *token, struct statement *s
static struct token *parse_goto_statement(struct token *token, struct statement
static struct token *parse_context_statement(struct token *token, struct stateme
static struct token *parse_range_statement(struct token *token, struct statement
static struct token *parse_asm_statement(struct token *token, struct statement *
static struct token *toplevel_asm_declaration(struct token *token, struct symbol
static struct token *parse_static_assert(struct token *token, struct symbol_list

typedef struct token *attr_t(struct token *, struct symbol *,
                             struct decl_state *);

static attr_t
        attribute_packed, attribute_aligned, attribute_modifier,
        attribute_bitwise,
        attribute_address_space, attribute_context,
        attribute_designated_init,
        attribute_transparent_union, ignore_attribute,
        attribute_mode, attribute_force;

typedef struct symbol *to_mode_t(struct symbol *);

static to_mode_t
        to_QI_mode, to_HI_mode, to_SI_mode, to_DI_mode, to_TI_mode, to_word_mode

enum {
        Set_T = 1,
        Set_S = 2,
        Set_Char = 4,
        Set_Int = 8,
        Set_Double = 16,
        Set_Float = 32,
        Set_Signed = 64,
        Set_Unsigned = 128,
        Set_Short = 256,
        Set_Long = 512,
        Set_Vlong = 1024,
        Set_Int128 = 2048,
        Set_Any = Set_T | Set_Short | Set_Long | Set_Signed | Set_Unsigned
};

enum {
        CInt = 0, CSInt, CUInt, CReal, CChar, CSChar, CUChar,
};

enum {
        SNone = 0, STypedef, SAuto, SRegister, SExtern, SStatic, SForced, SMax,
};

static struct symbol_op typedef_op = {
        .type = KW_MODIFIER,
        .declarator = typedef_specifier,
};

static struct symbol_op inline_op = {
        .type = KW_MODIFIER,
        .declarator = inline_specifier,
};
```

```
128 static declarator_t noreturn_specifier;
129 static struct symbol_op noreturn_op = {
130         .type = KW_MODIFIER,
131         .declarator = noreturn_specifier,
132 };

134 static declarator_t alignas_specifier;
135 static struct symbol_op alignas_op = {
136         .type = KW_MODIFIER,
137         .declarator = alignas_specifier,
138 };

140 static struct symbol_op auto_op = {
141         .type = KW_MODIFIER,
142         .declarator = auto_specifier,
143 };

145 static struct symbol_op register_op = {
146         .type = KW_MODIFIER,
147         .declarator = register_specifier,
148 };

150 static struct symbol_op static_op = {
151         .type = KW_MODIFIER,
152         .declarator = static_specifier,
153 };

155 static struct symbol_op extern_op = {
156         .type = KW_MODIFIER,
157         .declarator = extern_specifier,
158 };

160 static struct symbol_op thread_op = {
161         .type = KW_MODIFIER,
162         .declarator = thread_specifier,
163 };

165 static struct symbol_op const_op = {
166         .type = KW_QUALIFIER,
167         .declarator = const_qualifier,
168 };

170 static struct symbol_op volatile_op = {
171         .type = KW_QUALIFIER,
172         .declarator = volatile_qualifier,
173 };

175 static struct symbol_op restrict_op = {
176         .type = KW_QUALIFIER,
177 };

179 static struct symbol_op typeof_op = {
180         .type = KW_SPECIFIER,
181         .declarator = typeof_specifier,
182         .test = Set_Any,
183         .set = Set_S|Set_T,
184 };

186 static struct symbol_op attribute_op = {
187         .type = KW_ATTRIBUTE,
188         .declarator = attribute_specifier,
189 };

191 static struct symbol_op struct_op = {
192         .type = KW_SPECIFIER,
```



```
193         .declarator = struct_specifier,
194         .test = Set_Any,
195         .set = Set_S|Set_T,
196 };

198 static struct symbol_op union_op = {
199         .type = KW_SPECIFIER,
200         .declarator = union_specifier,
201         .test = Set_Any,
202         .set = Set_S|Set_T,
203 };

205 static struct symbol_op enum_op = {
206         .type = KW_SPECIFIER,
207         .declarator = enum_specifier,
208         .test = Set_Any,
209         .set = Set_S|Set_T,
210 };

212 static struct symbol_op spec_op = {
213         .type = KW_SPECIFIER | KW_EXACT,
214         .test = Set_Any,
215         .set = Set_S|Set_T,
216 };

218 static struct symbol_op char_op = {
219         .type = KW_SPECIFIER,
220         .test = Set_T|Set_Long|Set_Short,
221         .set = Set_T|Set_Char,
222         .class = CChar,
223 };

225 static struct symbol_op int_op = {
226         .type = KW_SPECIFIER,
227         .test = Set_T,
228         .set = Set_T|Set_Int,
229 };

231 static struct symbol_op double_op = {
232         .type = KW_SPECIFIER,
233         .test = Set_T|Set_Signed|Set_Unsigned|Set_Short|Set_Vlong,
234         .set = Set_T|Set_Double,
235         .class = CReal,
236 };

238 /* FIXME: this is not even slightly right. */
239 static struct symbol_op complex_op = {
240         .type = KW_SPECIFIER,
241         .test = 0, //Set_T|Set_Signed|Set_Unsigned|Set_Short|Set_Vlong,
242         .set = 0, //Set_Double, //Set_T,Set_Double,
243         .class = CReal,
244 };

246 static struct symbol_op float_op = {
247         .type = KW_SPECIFIER | KW_SHORT,
248         .test = Set_T|Set_Signed|Set_Unsigned|Set_Short|Set_Long,
249         .set = Set_T|Set_Float,
250         .class = CReal,
251 };

253 static struct symbol_op short_op = {
254         .type = KW_SPECIFIER | KW_SHORT,
255         .test = Set_S|Set_Char|Set_Float|Set_Double|Set_Long|Set_Short,
256         .set = Set_Short,
257 };
```

```
259 static struct symbol_op signed_op = {
260         .type = KW_SPECIFIER,
261         .test = Set_S|Set_Float|Set_Double|Set_Signed|Set_Unsigned,
262         .set = Set_Signed,
263         .class = CSInt,
264 };

266 static struct symbol_op unsigned_op = {
267         .type = KW_SPECIFIER,
268         .test = Set_S|Set_Float|Set_Double|Set_Signed|Set_Unsigned,
269         .set = Set_Unsigned,
270         .class = CUInt,
271 };

273 static struct symbol_op long_op = {
274         .type = KW_SPECIFIER | KW_LONG,
275         .test = Set_S|Set_Char|Set_Float|Set_Short|Set_Vlong,
276         .set = Set_Long,
277 };

279 static struct symbol_op int128_op = {
280         .type = KW_SPECIFIER | KW_LONG,
281         .test = Set_S|Set_T|Set_Char|Set_Short|Set_Int|Set_Float|Set_Double|Set_
282         .set =  Set_T|Set_Int128,
283 };

285 static struct symbol_op if_op = {
286         .statement = parse_if_statement,
287 };

289 static struct symbol_op return_op = {
290         .statement = parse_return_statement,
291 };

293 static struct symbol_op loop_iter_op = {
294         .statement = parse_loop_iterator,
295 };

297 static struct symbol_op default_op = {
298         .statement = parse_default_statement,
299 };

301 static struct symbol_op case_op = {
302         .statement = parse_case_statement,
303 };

305 static struct symbol_op switch_op = {
306         .statement = parse_switch_statement,
307 };

309 static struct symbol_op for_op = {
310         .statement = parse_for_statement,
311 };

313 static struct symbol_op while_op = {
314         .statement = parse_while_statement,
315 };

317 static struct symbol_op do_op = {
318         .statement = parse_do_statement,
319 };

321 static struct symbol_op goto_op = {
322         .statement = parse_goto_statement,
323 };
```



```
325 static struct symbol_op __context___op = {
326         .statement = parse_context_statement,
327 };

329 static struct symbol_op range_op = {
330         .statement = parse_range_statement,
331 };

333 static struct symbol_op asm_op = {
334         .type = KW_ASM,
335         .declarator = parse_asm_declarator,
336         .statement = parse_asm_statement,
337         .toplevel = toplevel_asm_declaration,
338 };

340 static struct symbol_op static_assert_op = {
341         .toplevel = parse_static_assert,
342 };

344 static struct symbol_op packed_op = {
345         .attribute = attribute_packed,
346 };

348 static struct symbol_op aligned_op = {
349         .attribute = attribute_aligned,
350 };

352 static struct symbol_op attr_mod_op = {
353         .attribute = attribute_modifier,
354 };

356 static struct symbol_op attr_bitwise_op = {
357         .attribute = attribute_bitwise,
358 };

360 static struct symbol_op attr_force_op = {
361         .attribute = attribute_force,
362 };

364 static struct symbol_op address_space_op = {
365         .attribute = attribute_address_space,
366 };

368 static struct symbol_op mode_op = {
369         .attribute = attribute_mode,
370 };

372 static struct symbol_op context_op = {
373         .attribute = attribute_context,
374 };

376 static struct symbol_op designated_init_op = {
377         .attribute = attribute_designated_init,
378 };

380 static struct symbol_op transparent_union_op = {
381         .attribute = attribute_transparent_union,
382 };

384 static struct symbol_op ignore_attr_op = {
385         .attribute = ignore_attribute,
386 };

388 static struct symbol_op mode_QI_op = {
389         .type = KW_MODE,
390         .to_mode = to_QI_mode
```

```
};

static struct symbol_op mode_HI_op = {
        .type = KW_MODE,
        .to_mode = to_HI_mode
};

static struct symbol_op mode_SI_op = {
        .type = KW_MODE,
        .to_mode = to_SI_mode
};

static struct symbol_op mode_DI_op = {
        .type = KW_MODE,
        .to_mode = to_DI_mode
};

static struct symbol_op mode_TI_op = {
        .type = KW_MODE,
        .to_mode = to_TI_mode
};

static struct symbol_op mode_word_op = {
        .type = KW_MODE,
        .to_mode = to_word_mode
};

/* Using NS_TYPEDEF will also make the keyword a reserved one */
static struct init_keyword {
        const char *name;
        enum namespace ns;
        unsigned long modifiers;
        struct symbol_op *op;
        struct symbol *type;
} keyword_table[] = {
        /* Type qualifiers */
        { "const",      NS_TYPEDEF, .op = &const_op },
        { "__const",    NS_TYPEDEF, .op = &const_op },
        { "__const__",  NS_TYPEDEF, .op = &const_op },
        { "volatile",   NS_TYPEDEF, .op = &volatile_op },
        { "__volatile",         NS_TYPEDEF, .op = &volatile_op },
        { "__volatile__",       NS_TYPEDEF, .op = &volatile_op },

        /* Typedef.. */
        { "typedef",    NS_TYPEDEF, .op = &typedef_op },

        /* Type specifiers */
        { "void",       NS_TYPEDEF, .type = &void_ctype, .op = &spec_op},
        { "char",       NS_TYPEDEF, .op = &char_op },
        { "short",      NS_TYPEDEF, .op = &short_op },
        { "int",        NS_TYPEDEF, .op = &int_op },
        { "long",       NS_TYPEDEF, .op = &long_op },
        { "float",      NS_TYPEDEF, .op = &float_op },
        { "double",     NS_TYPEDEF, .op = &double_op },
        { "signed",     NS_TYPEDEF, .op = &signed_op },
        { "__signed",   NS_TYPEDEF, .op = &signed_op },
        { "__signed__", NS_TYPEDEF, .op = &signed_op },
        { "unsigned",   NS_TYPEDEF, .op = &unsigned_op },
        { "__int128",   NS_TYPEDEF, .op = &int128_op },
        { "_Bool",      NS_TYPEDEF, .type = &bool_ctype, .op = &spec_op },
        { "_Complex",   NS_TYPEDEF, .op = &complex_op },

        /* Predeclared types */
        { "__builtin_va_list", NS_TYPEDEF, .type = &ptr_ctype, .op = &spec_op },
        { "__builtin_ms_va_list", NS_TYPEDEF, .type = &ptr_ctype, .op = &spec_op
        { "__int128_t", NS_TYPEDEF, .type = &lllong_ctype, .op = &spec_op },
```



```
        { "__uint128_t",NS_TYPEDEF, .type = &ulllong_ctype, .op = &spec_op },

        /* Extended types */
        { "typeof",     NS_TYPEDEF, .op = &typeof_op },
        { "__typeof",   NS_TYPEDEF, .op = &typeof_op },
        { "__typeof__", NS_TYPEDEF, .op = &typeof_op },

        { "__attribute",   NS_TYPEDEF, .op = &attribute_op },
        { "__attribute__", NS_TYPEDEF, .op = &attribute_op },

        { "struct",     NS_TYPEDEF, .op = &struct_op },
        { "union",      NS_TYPEDEF, .op = &union_op },
        { "enum",       NS_TYPEDEF, .op = &enum_op },

        { "inline",     NS_TYPEDEF, .op = &inline_op },
        { "__inline",   NS_TYPEDEF, .op = &inline_op },
        { "__inline__", NS_TYPEDEF, .op = &inline_op },

        { "_Noreturn",  NS_TYPEDEF, .op = &noreturn_op },

        { "_Alignas",   NS_TYPEDEF, .op = &alignas_op },

        /* Ignored for now.. */
        { "restrict",   NS_TYPEDEF, .op = &restrict_op},
        { "__restrict", NS_TYPEDEF, .op = &restrict_op},
        { "__restrict__",       NS_TYPEDEF, .op = &restrict_op},

        /* Static assertion */
        { "_Static_assert", NS_KEYWORD, .op = &static_assert_op },

        /* Storage class */
        { "auto",       NS_TYPEDEF, .op = &auto_op },
        { "register",   NS_TYPEDEF, .op = &register_op },
        { "static",     NS_TYPEDEF, .op = &static_op },
        { "extern",     NS_TYPEDEF, .op = &extern_op },
        { "__thread",   NS_TYPEDEF, .op = &thread_op },
        { "_Thread_local",      NS_TYPEDEF, .op = &thread_op },

        /* Statement */
        { "if",         NS_KEYWORD, .op = &if_op },
        { "return",     NS_KEYWORD, .op = &return_op },
        { "break",      NS_KEYWORD, .op = &loop_iter_op },
        { "continue",   NS_KEYWORD, .op = &loop_iter_op },
        { "default",    NS_KEYWORD, .op = &default_op },
        { "case",       NS_KEYWORD, .op = &case_op },
        { "switch",     NS_KEYWORD, .op = &switch_op },
        { "for",        NS_KEYWORD, .op = &for_op },
        { "while",      NS_KEYWORD, .op = &while_op },
        { "do",         NS_KEYWORD, .op = &do_op },
        { "goto",       NS_KEYWORD, .op = &goto_op },
        { "__context__",NS_KEYWORD, .op = &__context__op },
        { "__range__",  NS_KEYWORD, .op = &range_op },
        { "asm",        NS_KEYWORD, .op = &asm_op },
        { "__asm",      NS_KEYWORD, .op = &asm_op },
        { "__asm__",    NS_KEYWORD, .op = &asm_op },

        /* Attribute */
        { "packed",     NS_KEYWORD, .op = &packed_op },
        { "__packed__", NS_KEYWORD, .op = &packed_op },
        { "aligned",    NS_KEYWORD, .op = &aligned_op },
        { "__aligned__",NS_KEYWORD, .op = &aligned_op },
        { "nocast",     NS_KEYWORD,     MOD_NOCAST,     .op = &attr_mod_op },
        { "noderef",    NS_KEYWORD,     MOD_NODEREF,    .op = &attr_mod_op },
        { "safe",       NS_KEYWORD,     MOD_SAFE,       .op = &attr_mod_op },
        { "force",      NS_KEYWORD,     .op = &attr_force_op },
        { "bitwise",    NS_KEYWORD,     MOD_BITWISE,    .op = &attr_bitwise_op }
```

```
523         { "__bitwise__",NS_KEYWORD,     MOD_BITWISE,    .op = &attr_bitwise_op }
524         { "address_space",NS_KEYWORD,   .op = &address_space_op },
525         { "mode",       NS_KEYWORD,     .op = &mode_op },
526         { "context",    NS_KEYWORD,     .op = &context_op },
527         { "designated_init",    NS_KEYWORD,     .op = &designated_init_op },
528         { "__transparent_union__",      NS_KEYWORD,     .op = &transparent_union
529         { "noreturn",   NS_KEYWORD,     MOD_NORETURN,   .op = &attr_mod_op },
530         { "__noreturn__",       NS_KEYWORD,     MOD_NORETURN,   .op = &attr_mod_
531         { "pure",       NS_KEYWORD,     MOD_PURE,       .op = &attr_mod_op },
532         {"__pure__",    NS_KEYWORD,     MOD_PURE,       .op = &attr_mod_op },
533         {"const",       NS_KEYWORD,     MOD_PURE,       .op = &attr_mod_op },
534         {"__const",     NS_KEYWORD,     MOD_PURE,       .op = &attr_mod_op },
535         {"__const__",   NS_KEYWORD,     MOD_PURE,       .op = &attr_mod_op },

537         { "__mode__",   NS_KEYWORD,     .op = &mode_op },
538         { "QI",         NS_KEYWORD,     MOD_CHAR,       .op = &mode_QI_op },
539         { "__QI__",     NS_KEYWORD,     MOD_CHAR,       .op = &mode_QI_op },
540         { "HI",         NS_KEYWORD,     MOD_SHORT,      .op = &mode_HI_op },
541         { "__HI__",     NS_KEYWORD,     MOD_SHORT,      .op = &mode_HI_op },
542         { "SI",         NS_KEYWORD,                     .op = &mode_SI_op },
543         { "__SI__",     NS_KEYWORD,                     .op = &mode_SI_op },
544         { "DI",         NS_KEYWORD,     MOD_LONGLONG,   .op = &mode_DI_op },
545         { "__DI__",     NS_KEYWORD,     MOD_LONGLONG,   .op = &mode_DI_op },
546         { "TI",         NS_KEYWORD,     MOD_LONGLONGLONG,       .op = &mode_TI_o
547         { "__TI__",     NS_KEYWORD,     MOD_LONGLONGLONG,       .op = &mode_TI_o
548         { "word",       NS_KEYWORD,     MOD_LONG,       .op = &mode_word_op },
549         { "__word__",   NS_KEYWORD,     MOD_LONG,       .op = &mode_word_op },
550 };


553 static const char *ignored_attributes[] = {

555 #define GCC_ATTR(x)             \
556         STRINGIFY(x),           \
557         STRINGIFY(__##x##__),

559 #include "gcc-attr-list.h"

561 #undef GCC_ATTR

563         "bounded",
564         "__bounded__",
565         "__noclone",
566         "__nonnull",
567         "__nothrow",
568 };


571 void init_parser(int stream)
572 {
573         int i;
574         for (i = 0; i < ARRAY_SIZE(keyword_table); i++) {
575                 struct init_keyword *ptr = keyword_table + i;
576                 struct symbol *sym = create_symbol(stream, ptr->name, SYM_KEYWOR
577                 sym->ident->keyword = 1;
578                 if (ptr->ns == NS_TYPEDEF)
579                         sym->ident->reserved = 1;
580                 sym->ctype.modifiers = ptr->modifiers;
581                 sym->ctype.base_type = ptr->type;
582                 sym->op = ptr->op;
583         }

585         for (i = 0; i < ARRAY_SIZE(ignored_attributes); i++) {
586                 const char * name = ignored_attributes[i];
587                 struct symbol *sym = create_symbol(stream, name, SYM_KEYWORD,
588                                                    NS_KEYWORD);
```



```
589                 if (!sym->op) {
590                         sym->ident->keyword = 1;
591                         sym->op = &ignore_attr_op;
592                 }
593         }
594 }


597 // Add a symbol to the list of function-local symbols
598 static void fn_local_symbol(struct symbol *sym)
599 {
600         if (function_symbol_list)
601                 add_symbol(function_symbol_list, sym);
602 }

604 static int SENTINEL_ATTR match_idents(struct token *token, ...)
605 {
606         va_list args;
607         struct ident * next;

609         if (token_type(token) != TOKEN_IDENT)
610                 return 0;

612         va_start(args, token);
613         do {
614                 next = va_arg(args, struct ident *);
615         } while (next && token->ident != next);
616         va_end(args);

618         return next && token->ident == next;
619 }


622 struct statement *alloc_statement(struct position pos, int type)
623 {
624         struct statement *stmt = __alloc_statement(0);
625         stmt->type = type;
626         stmt->pos = pos;
627         return stmt;
628 }

630 static struct token *struct_declaration_list(struct token *token, struct symbol_

632 static void apply_ctype(struct position pos, struct ctype *thistype, struct ctyp

634 static void apply_modifiers(struct position pos, struct decl_state *ctx)
635 {
636         struct symbol *ctype;
637         if (!ctx->mode)
638                 return;
639         ctype = ctx->mode->to_mode(ctx->ctype.base_type);
640         if (!ctype)
641                 sparse_error(pos, "don't know how to apply mode to %s",
642                                 show_typename(ctx->ctype.base_type));
643         else
644                 ctx->ctype.base_type = ctype;
645
646 }

648 static struct symbol * alloc_indirect_symbol(struct position pos, struct ctype *
649 {
650         struct symbol *sym = alloc_symbol(pos, type);

652         sym->ctype.base_type = ctype->base_type;
653         sym->ctype.modifiers = ctype->modifiers;
```

```
655         ctype->base_type = sym;
656         ctype->modifiers = 0;
657         return sym;
658 }

660 /*
661  * NOTE! NS_LABEL is not just a different namespace,
662  * it also ends up using function scope instead of the
663  * regular symbol scope.
664  */
665 struct symbol *label_symbol(struct token *token)
666 {
667         struct symbol *sym = lookup_symbol(token->ident, NS_LABEL);
668         if (!sym) {
669                 sym = alloc_symbol(token->pos, SYM_LABEL);
670                 bind_symbol(sym, token->ident, NS_LABEL);
671                 fn_local_symbol(sym);
672         }
673         return sym;
674 }

676 static struct token *struct_union_enum_specifier(enum type type,
677         struct token *token, struct decl_state *ctx,
678         struct token *(*parse)(struct token *, struct symbol *))
679 {
680         struct symbol *sym;
681         struct position *repos;

683         token = handle_attributes(token, ctx, KW_ATTRIBUTE);
684         if (token_type(token) == TOKEN_IDENT) {
685                 sym = lookup_symbol(token->ident, NS_STRUCT);
686                 if (!sym ||
687                     (is_outer_scope(sym->scope) &&
688                      (match_op(token->next,';') || match_op(token->next,'{'))))
689                         // Either a new symbol, or else an out-of-scope
690                         // symbol being redefined.
691                         sym = alloc_symbol(token->pos, type);
692                         bind_symbol(sym, token->ident, NS_STRUCT);
693                 }
694                 if (sym->type != type)
695                         error_die(token->pos, "invalid tag applied to %s", show_
696                 ctx->ctype.base_type = sym;
697                 repos = &token->pos;
698                 token = token->next;
699                 if (match_op(token, '{')) {
700                         struct decl_state attr = { .ctype.base_type = sym, };

702                         // The following test is actually wrong for empty
703                         // structs, but (1) they are not C99, (2) gcc does
704                         // the same thing, and (3) it's easier.
705                         if (sym->symbol_list)
706                                 error_die(token->pos, "redefinition of %s", show
707                         sym->pos = *repos;
708                         token = parse(token->next, sym);
709                         token = expect(token, '}', "at end of struct-union-enum-

711                         token = handle_attributes(token, &attr, KW_ATTRIBUTE);
712                         apply_ctype(token->pos, &attr.ctype, &sym->ctype);

714                         // Mark the structure as needing re-examination
715                         sym->examined = 0;
716                         sym->endpos = token->pos;
717                 }
718                 return token;
719         }
```



```
721         // private struct/union/enum type
722         if (!match_op(token, '{')) {
723                 sparse_error(token->pos, "expected declaration");
724                 ctx->ctype.base_type = &bad_ctype;
725                 return token;
726         }

728         sym = alloc_symbol(token->pos, type);
729         token = parse(token->next, sym);
730         ctx->ctype.base_type = sym;
731         token =  expect(token, '}', "at end of specifier");
732         sym->endpos = token->pos;

734         return token;
735 }

737 static struct token *parse_struct_declaration(struct token *token, struct symbol
738 {
739         struct symbol *field, *last = NULL;
740         struct token *res;
741         res = struct_declaration_list(token, &sym->symbol_list);
742         FOR_EACH_PTR(sym->symbol_list, field) {
743                 if (!field->ident) {
744                         struct symbol *base = field->ctype.base_type;
745                         if (base && base->type == SYM_BITFIELD)
746                                 continue;
747                 }
748                 if (last)
749                         last->next_subobject = field;
750                 last = field;
751         } END_FOR_EACH_PTR(field);
752         return res;
753 }

755 static struct token *parse_union_declaration(struct token *token, struct symbol
756 {
757         return struct_declaration_list(token, &sym->symbol_list);
758 }

760 static struct token *struct_specifier(struct token *token, struct decl_state *ct
761 {
762         return struct_union_enum_specifier(SYM_STRUCT, token, ctx, parse_struct_
763 }

765 static struct token *union_specifier(struct token *token, struct decl_state *ctx
766 {
767         return struct_union_enum_specifier(SYM_UNION, token, ctx, parse_union_de
768 }


771 typedef struct {
772         int x;
773         unsigned long long y;
774 } Num;

776 static void upper_boundary(Num *n, Num *v)
777 {
778         if (n->x > v->x)
779                 return;
780         if (n->x < v->x) {
781                 *n = *v;
782                 return;
783         }
784         if (n->y < v->y)
785                 n->y = v->y;
786 }
```

```
788 static void lower_boundary(Num *n, Num *v)
789 {
790         if (n->x < v->x)
791                 return;
792         if (n->x > v->x) {
793                 *n = *v;
794                 return;
795         }
796         if (n->y > v->y)
797                 n->y = v->y;
798 }

800 static int type_is_ok(struct symbol *type, Num *upper, Num *lower)
801 {
802         int shift = type->bit_size;
803         int is_unsigned = type->ctype.modifiers & MOD_UNSIGNED;

805         if (!is_unsigned)
806                 shift--;
807         if (upper->x == 0 && upper->y >> shift)
808                 return 0;
809         if (lower->x == 0 || (!is_unsigned && (~lower->y >> shift) == 0))
810                 return 1;
811         return 0;
812 }

814 static struct symbol *bigger_enum_type(struct symbol *s1, struct symbol *s2)
815 {
816         if (s1->bit_size < s2->bit_size) {
817                 s1 = s2;
818         } else if (s1->bit_size == s2->bit_size) {
819                 if (s2->ctype.modifiers & MOD_UNSIGNED)
820                         s1 = s2;
821         }
822         if (s1->bit_size < bits_in_int)
823                 return &int_ctype;
824         return s1;
825 }

827 static void cast_enum_list(struct symbol_list *list, struct symbol *base_type)
828 {
829         struct symbol *sym;

831         FOR_EACH_PTR(list, sym) {
832                 struct expression *expr = sym->initializer;
833                 struct symbol *ctype;
834                 if (expr->type != EXPR_VALUE)
835                         continue;
836                 ctype = expr->ctype;
837                 if (ctype->bit_size == base_type->bit_size)
838                         continue;
839                 cast_value(expr, base_type, expr, ctype);
840         } END_FOR_EACH_PTR(sym);
841 }

843 static struct token *parse_enum_declaration(struct token *token, struct symbol *
844 {
845         unsigned long long lastval = 0;
846         struct symbol *ctype = NULL, *base_type = NULL;
847         Num upper = {-1, 0}, lower = {1, 0};

849         parent->examined = 1;
850         parent->ctype.base_type = &int_ctype;
851         while (token_type(token) == TOKEN_IDENT) {
852                 struct expression *expr = NULL;
```



```
853                 struct token *next = token->next;
854                 struct symbol *sym;

856                 if (match_op(next, '=')) {
857                         next = constant_expression(next->next, &expr);
858                         lastval = get_expression_value(expr);
859                         ctype = &void_ctype;
860                         if (expr && expr->ctype)
861                                 ctype = expr->ctype;
862                 } else if (!ctype) {
863                         ctype = &int_ctype;
864                 } else if (is_int_type(ctype)) {
865                         lastval++;
866                 } else {
867                         error_die(token->pos, "can't increment the last enum mem
868                 }

870                 if (!expr) {
871                         expr = alloc_expression(token->pos, EXPR_VALUE);
872                         expr->value = lastval;
873                         expr->ctype = ctype;
874                 }

876                 sym = alloc_symbol(token->pos, SYM_NODE);
877                 bind_symbol(sym, token->ident, NS_SYMBOL);
878                 sym->ctype.modifiers &= ~MOD_ADDRESSABLE;
879                 sym->initializer = expr;
880                 sym->enum_member = 1;
881                 sym->ctype.base_type = parent;
882                 add_ptr_list(&parent->symbol_list, sym);

884                 if (base_type != &bad_ctype) {
885                         if (ctype->type == SYM_NODE)
886                                 ctype = ctype->ctype.base_type;
887                         if (ctype->type == SYM_ENUM) {
888                                 if (ctype == parent)
889                                         ctype = base_type;
890                                 else
891                                         ctype = ctype->ctype.base_type;
892                         }
893                         /*
894                          * base_type rules:
895                          *  - if all enums are of the same type, then
896                          *    the base_type is that type (two first
897                          *    cases)
898                          *  - if enums are of different types, they
899                          *    all have to be integer types, and the
900                          *    base type is at least "int_ctype".
901                          *  - otherwise the base_type is "bad_ctype".
902                          */
903                         if (!base_type) {
904                                 base_type = ctype;
905                         } else if (ctype == base_type) {
906                                 /* nothing */
907                         } else if (is_int_type(base_type) && is_int_type(ctype))
908                                 base_type = bigger_enum_type(base_type, ctype);
909                         } else
910                                 base_type = &bad_ctype;
911                         parent->ctype.base_type = base_type;
912                 }
913                 if (is_int_type(base_type)) {
914                         Num v = {.y = lastval};
915                         if (ctype->ctype.modifiers & MOD_UNSIGNED)
916                                 v.x = 0;
917                         else if ((long long)lastval >= 0)
918                                 v.x = 0;
```

```
919                         else
920                                 v.x = -1;
921                         upper_boundary(&upper, &v);
922                         lower_boundary(&lower, &v);
923                 }
924                 token = next;

926                 sym->endpos = token->pos;

928                 if (!match_op(token, ','))
929                         break;
930                 token = token->next;
931         }
932         if (!base_type) {
933                 sparse_error(token->pos, "bad enum definition");
934                 base_type = &bad_ctype;
935         }
936         else if (!is_int_type(base_type))
937                 base_type = base_type;
938         else if (type_is_ok(base_type, &upper, &lower))
939                 base_type = base_type;
940         else if (type_is_ok(&int_ctype, &upper, &lower))
941                 base_type = &int_ctype;
942         else if (type_is_ok(&uint_ctype, &upper, &lower))
943                 base_type = &uint_ctype;
944         else if (type_is_ok(&long_ctype, &upper, &lower))
945                 base_type = &long_ctype;
946         else if (type_is_ok(&ulong_ctype, &upper, &lower))
947                 base_type = &ulong_ctype;
948         else if (type_is_ok(&llong_ctype, &upper, &lower))
949                 base_type = &llong_ctype;
950         else if (type_is_ok(&ullong_ctype, &upper, &lower))
951                 base_type = &ullong_ctype;
952         else
953                 base_type = &bad_ctype;
954         parent->ctype.base_type = base_type;
955         parent->ctype.modifiers |= (base_type->ctype.modifiers & MOD_UNSIGNED);
956         parent->examined = 0;

958         cast_enum_list(parent->symbol_list, base_type);

960         return token;
961 }

963 static struct token *enum_specifier(struct token *token, struct decl_state *ctx)
964 {
965         struct token *ret = struct_union_enum_specifier(SYM_ENUM, token, ctx, pa
966         struct ctype *ctype = &ctx->ctype.base_type->ctype;

968         if (!ctype->base_type)
969                 ctype->base_type = &incomplete_ctype;

971         return ret;
972 }

974 static struct token *typeof_specifier(struct token *token, struct decl_state *ct
975 {
976         struct symbol *sym;

978         if (!match_op(token, '(')) {
979                 sparse_error(token->pos, "expected '(' after typeof");
980                 return token;
981         }
982         if (lookup_type(token->next)) {
983                 token = typename(token->next, &sym, NULL);
984                 ctx->ctype.base_type = sym->ctype.base_type;
```



```
985                 apply_ctype(token->pos, &sym->ctype, &ctx->ctype);
986         } else {
987                 struct symbol *typeof_sym = alloc_symbol(token->pos, SYM_TYPEOF)
988                 token = parse_expression(token->next, &typeof_sym->initializer);

990                 typeof_sym->endpos = token->pos;
991                 if (!typeof_sym->initializer) {
992                         sparse_error(token->pos, "expected expression after the
993                         typeof_sym = &bad_ctype;
994                 }
995                 ctx->ctype.base_type = typeof_sym;
996         }
997         return expect(token, ')', "after typeof");
998 }

1000 static struct token *ignore_attribute(struct token *token, struct symbol *attr,
1001 {
1002         struct expression *expr = NULL;
1003         if (match_op(token, '('))
1004                 token = parens_expression(token, &expr, "in attribute");
1005         return token;
1006 }

1008 static struct token *attribute_packed(struct token *token, struct symbol *attr,
1009 {
1010         if (!ctx->ctype.alignment)
1011                 ctx->ctype.alignment = 1;
1012         return token;
1013 }

1015 static struct token *attribute_aligned(struct token *token, struct symbol *attr,
1016 {
1017         int alignment = max_alignment;
1018         struct expression *expr = NULL;

1020         if (match_op(token, '(')) {
1021                 token = parens_expression(token, &expr, "in attribute");
1022                 if (expr)
1023                         alignment = const_expression_value(expr);
1024         }
1025         if (alignment & (alignment-1)) {
1026                 warning(token->pos, "I don't like non-power-of-2 alignments");
1027                 return token;
1028         } else if (alignment > ctx->ctype.alignment)
1029                 ctx->ctype.alignment = alignment;
1030         return token;
1031 }

1033 static void apply_qualifier(struct position *pos, struct ctype *ctx, unsigned lo
1034 {
1035         if (ctx->modifiers & qual)
1036                 warning(*pos, "duplicate %s", modifier_string(qual));
1037         ctx->modifiers |= qual;
1038 }

1040 static struct token *attribute_modifier(struct token *token, struct symbol *attr
1041 {
1042         apply_qualifier(&token->pos, &ctx->ctype, attr->ctype.modifiers);
1043         return token;
1044 }

1046 static struct token *attribute_bitwise(struct token *token, struct symbol *attr,
1047 {
1048         if (Wbitwise)
1049                 attribute_modifier(token, attr, ctx);
1050         return token;
```

```
}

static struct token *attribute_address_space(struct token *token, struct symbol
{
        struct expression *expr = NULL;
        int as;
        token = expect(token, '(', "after address_space attribute");
        token = conditional_expression(token, &expr);
        if (expr) {
                as = const_expression_value(expr);
                if (Waddress_space && as)
                        ctx->ctype.as = as;
        }
        token = expect(token, ')', "after address_space attribute");
        return token;
}

static struct symbol *to_QI_mode(struct symbol *ctype)
{
        if (ctype->ctype.base_type != &int_type)
                return NULL;
        if (ctype == &char_ctype)
                return ctype;
        return ctype->ctype.modifiers & MOD_UNSIGNED ? &uchar_ctype
                                                     : &schar_ctype;
}

static struct symbol *to_HI_mode(struct symbol *ctype)
{
        if (ctype->ctype.base_type != &int_type)
                return NULL;
        return ctype->ctype.modifiers & MOD_UNSIGNED ? &ushort_ctype
                                                     : &sshort_ctype;
}

static struct symbol *to_SI_mode(struct symbol *ctype)
{
        if (ctype->ctype.base_type != &int_type)
                return NULL;
        return ctype->ctype.modifiers & MOD_UNSIGNED ? &uint_ctype
                                                     : &sint_ctype;
}

static struct symbol *to_DI_mode(struct symbol *ctype)
{
        if (ctype->ctype.base_type != &int_type)
                return NULL;
        return ctype->ctype.modifiers & MOD_UNSIGNED ? &ullong_ctype
                                                     : &sllong_ctype;
}

static struct symbol *to_TI_mode(struct symbol *ctype)
{
        if (ctype->ctype.base_type != &int_type)
                return NULL;
        return ctype->ctype.modifiers & MOD_UNSIGNED ? &ulllong_ctype
                                                     : &slllong_ctype;
}

static struct symbol *to_word_mode(struct symbol *ctype)
{
        if (ctype->ctype.base_type != &int_type)
                return NULL;
        return ctype->ctype.modifiers & MOD_UNSIGNED ? &ulong_ctype
                                                     : &slong_ctype;
}
```



```
static struct token *attribute_mode(struct token *token, struct symbol *attr, st
{
        token = expect(token, '(', "after mode attribute");
        if (token_type(token) == TOKEN_IDENT) {
                struct symbol *mode = lookup_keyword(token->ident, NS_KEYWORD);
                if (mode && mode->op->type == KW_MODE)
                        ctx->mode = mode->op;
                else
                        sparse_error(token->pos, "unknown mode attribute %s\n",
                token = token->next;
        } else
                sparse_error(token->pos, "expect attribute mode symbol\n");
        token = expect(token, ')', "after mode attribute");
        return token;
}

static struct token *attribute_context(struct token *token, struct symbol *attr,
{
        struct context *context = alloc_context();
        struct expression *args[3];
        int argc = 0;

        token = expect(token, '(', "after context attribute");
        while (!match_op(token, ')')) {
                struct expression *expr = NULL;
                token = conditional_expression(token, &expr);
                if (!expr)
                        break;
                if (argc < 3)
                        args[argc++] = expr;
                if (!match_op(token, ','))
                        break;
                token = token->next;
        }

        switch(argc) {
        case 0:
                sparse_error(token->pos, "expected context input/output values")
                break;
        case 1:
                context->in = get_expression_value(args[0]);
                break;
        case 2:
                context->in = get_expression_value(args[0]);
                context->out = get_expression_value(args[1]);
                break;
        case 3:
                context->context = args[0];
                context->in = get_expression_value(args[1]);
                context->out = get_expression_value(args[2]);
                break;
        }

        if (argc)
                add_ptr_list(&ctx->ctype.contexts, context);

        token = expect(token, ')', "after context attribute");
        return token;
}

static struct token *attribute_designated_init(struct token *token, struct symbo
{
        if (ctx->ctype.base_type && ctx->ctype.base_type->type == SYM_STRUCT)
                ctx->ctype.base_type->designated_init = 1;
        else
```

```
1183                 warning(token->pos, "attribute designated_init applied to non-st
1184         return token;
1185 }

1187 static struct token *attribute_transparent_union(struct token *token, struct sym
1188 {
1189         if (Wtransparent_union)
1190                 warning(token->pos, "attribute __transparent_union__");

1192         if (ctx->ctype.base_type && ctx->ctype.base_type->type == SYM_UNION)
1193                 ctx->ctype.base_type->transparent_union = 1;
1194         else
1195                 warning(token->pos, "attribute __transparent_union__ applied to
1196         return token;
1197 }

1199 static struct token *recover_unknown_attribute(struct token *token)
1200 {
1201         struct expression *expr = NULL;

1203         if (Wunknown_attribute)
1204                 warning(token->pos, "attribute '%s': unknown attribute", show_id
1205         token = token->next;
1206         if (match_op(token, '('))
1207                 token = parens_expression(token, &expr, "in attribute");
1208         return token;
1209 }

1211 static struct token *attribute_specifier(struct token *token, struct decl_state
1212 {
1213         token = expect(token, '(', "after attribute");
1214         token = expect(token, '(', "after attribute");

1216         for (;;) {
1217                 struct ident *attribute_name;
1218                 struct symbol *attr;

1220                 if (eof_token(token))
1221                         break;
1222                 if (match_op(token, ';'))
1223                         break;
1224                 if (token_type(token) != TOKEN_IDENT)
1225                         break;
1226                 attribute_name = token->ident;
1227                 attr = lookup_keyword(attribute_name, NS_KEYWORD);
1228                 if (attr && attr->op->attribute)
1229                         token = attr->op->attribute(token->next, attr, ctx);
1230                 else
1231                         token = recover_unknown_attribute(token);

1233                 if (!match_op(token, ','))
1234                         break;
1235                 token = token->next;
1236         }

1238         token = expect(token, ')', "after attribute");
1239         token = expect(token, ')', "after attribute");
1240         return token;
1241 }

1243 static const char *storage_class[] =
1244 {
1245         [STypedef] = "typedef",
1246         [SAuto] = "auto",
1247         [SExtern] = "extern",
1248         [SStatic] = "static",
```



```
1249         [SRegister] = "register",
1250         [SForced] = "[force]"
1251 };

1253 static unsigned long storage_modifiers(struct decl_state *ctx)
1254 {
1255         static unsigned long mod[SMax] =
1256         {
1257                 [SAuto] = MOD_AUTO,
1258                 [SExtern] = MOD_EXTERN,
1259                 [SStatic] = MOD_STATIC,
1260                 [SRegister] = MOD_REGISTER
1261         };
1262         return mod[ctx->storage_class] | (ctx->is_inline ? MOD_INLINE : 0)
1263                 | (ctx->is_tls ? MOD_TLS : 0);
1264 }

1266 static void set_storage_class(struct position *pos, struct decl_state *ctx, int
1267 {
1268         /* __thread can be used alone, or with extern or static */
1269         if (ctx->is_tls && (class != SStatic && class != SExtern)) {
1270                 sparse_error(*pos, "__thread can only be used alone, or with "
1271                                 "extern or static");
1272                 return;
1273         }

1275         if (!ctx->storage_class) {
1276                 ctx->storage_class = class;
1277                 return;
1278         }
1279         if (ctx->storage_class == class)
1280                 sparse_error(*pos, "duplicate %s", storage_class[class]);
1281         else
1282                 sparse_error(*pos, "multiple storage classes");
1283 }

1285 static struct token *typedef_specifier(struct token *next, struct decl_state *ct
1286 {
1287         set_storage_class(&next->pos, ctx, STypedef);
1288         return next;
1289 }

1291 static struct token *auto_specifier(struct token *next, struct decl_state *ctx)
1292 {
1293         set_storage_class(&next->pos, ctx, SAuto);
1294         return next;
1295 }

1297 static struct token *register_specifier(struct token *next, struct decl_state *c
1298 {
1299         set_storage_class(&next->pos, ctx, SRegister);
1300         return next;
1301 }

1303 static struct token *static_specifier(struct token *next, struct decl_state *ctx
1304 {
1305         set_storage_class(&next->pos, ctx, SStatic);
1306         return next;
1307 }

1309 static struct token *extern_specifier(struct token *next, struct decl_state *ctx
1310 {
1311         set_storage_class(&next->pos, ctx, SExtern);
1312         return next;
1313 }
```

```
1315 static struct token *thread_specifier(struct token *next, struct decl_state *ctx
1316 {
1317         /* This GCC extension can be used alone, or with extern or static */
1318         if (!ctx->storage_class || ctx->storage_class == SStatic
1319                         || ctx->storage_class == SExtern) {
1320                 ctx->is_tls = 1;
1321         } else {
1322                 sparse_error(next->pos, "__thread can only be used alone, or "
1323                                 "with extern or static");
1324         }
1325
1326         return next;
1327 }
1328
1329 static struct token *attribute_force(struct token *token, struct symbol *attr, s
1330 {
1331         set_storage_class(&token->pos, ctx, SForced);
1332         return token;
1333 }
1334
1335 static struct token *inline_specifier(struct token *next, struct decl_state *ctx
1336 {
1337         ctx->is_inline = 1;
1338         return next;
1339 }
1340
1341 static struct token *noreturn_specifier(struct token *next, struct decl_state *c
1342 {
1343         apply_qualifier(&next->pos, &ctx->ctype, MOD_NORETURN);
1344         return next;
1345 }
1346
1347 static struct token *alignas_specifier(struct token *token, struct decl_state *c
1348 {
1349         int alignment = 0;
1350
1351         if (!match_op(token, '(')) {
1352                 sparse_error(token->pos, "expected '(' after _Alignas");
1353                 return token;
1354         }
1355         if (lookup_type(token->next)) {
1356                 struct symbol *sym = NULL;
1357                 token = typename(token->next, &sym, NULL);
1358                 sym = examine_symbol_type(sym);
1359                 alignment = sym->ctype.alignment;
1360                 token = expect(token, ')', "after _Alignas(...");
1361         } else {
1362                 struct expression *expr = NULL;
1363                 token = parens_expression(token, &expr, "after _Alignas");
1364                 if (!expr)
1365                         return token;
1366                 alignment = const_expression_value(expr);
1367         }
1368
1369         if (alignment < 0) {
1370                 warning(token->pos, "non-positive alignment");
1371                 return token;
1372         }
1373         if (alignment & (alignment-1)) {
1374                 warning(token->pos, "non-power-of-2 alignment");
1375                 return token;
1376         }
1377         if (alignment > ctx->ctype.alignment)
1378                 ctx->ctype.alignment = alignment;
1379         return token;
1380 }
```



```
1382 static struct token *const_qualifier(struct token *next, struct decl_state *ctx)
1383 {
1384         apply_qualifier(&next->pos, &ctx->ctype, MOD_CONST);
1385         return next;
1386 }
1387
1388 static struct token *volatile_qualifier(struct token *next, struct decl_state *c
1389 {
1390         apply_qualifier(&next->pos, &ctx->ctype, MOD_VOLATILE);
1391         return next;
1392 }
1393
1394 static void apply_ctype(struct position pos, struct ctype *thistype, struct ctyp
1395 {
1396         unsigned long mod = thistype->modifiers;
1397
1398         if (mod)
1399                 apply_qualifier(&pos, ctype, mod);
1400
1401         /* Context */
1402         concat_ptr_list((struct ptr_list *)thistype->contexts,
1403                         (struct ptr_list **)&ctype->contexts);
1404
1405         /* Alignment */
1406         if (thistype->alignment > ctype->alignment)
1407                 ctype->alignment = thistype->alignment;
1408
1409         /* Address space */
1410         if (thistype->as)
1411                 ctype->as = thistype->as;
1412 }
1413
1414 static void specifier_conflict(struct position pos, int what, struct ident *new)
1415 {
1416         const char *old;
1417         if (what & (Set_S | Set_T))
1418                 goto Catch_all;
1419         if (what & Set_Char)
1420                 old = "char";
1421         else if (what & Set_Double)
1422                 old = "double";
1423         else if (what & Set_Float)
1424                 old = "float";
1425         else if (what & Set_Signed)
1426                 old = "signed";
1427         else if (what & Set_Unsigned)
1428                 old = "unsigned";
1429         else if (what & Set_Short)
1430                 old = "short";
1431         else if (what & Set_Long)
1432                 old = "long";
1433         else
1434                 old = "long long";
1435         sparse_error(pos, "impossible combination of type specifiers: %s %s",
1436                         old, show_ident(new));
1437         return;
1438
1439 Catch_all:
1440         sparse_error(pos, "two or more data types in declaration specifiers");
1441 }
1442
1443 static struct symbol * const int_types[] =
1444         {&short_ctype, &int_ctype, &long_ctype, &llong_ctype, &lllong_ctype};
1445 static struct symbol * const signed_types[] =
1446         {&sshort_ctype, &sint_ctype, &slong_ctype, &sllong_ctype,
```

```
1447          &slllong_ctype};
1448 static struct symbol * const unsigned_types[] =
1449         {&ushort_ctype, &uint_ctype, &ulong_ctype, &ullong_ctype,
1450          &ulllong_ctype};
1451 static struct symbol * const real_types[] =
1452         {&float_ctype, &double_ctype, &ldouble_ctype};
1453 static struct symbol * const char_types[] =
1454         {&char_ctype, &schar_ctype, &uchar_ctype};
1455 static struct symbol * const * const types[] = {
1456         int_types + 1, signed_types + 1, unsigned_types + 1,
1457         real_types + 1, char_types, char_types + 1, char_types + 2
1458 };

1460 struct symbol *ctype_integer(int size, int want_unsigned)
1461 {
1462         return types[want_unsigned ? CUInt : CInt][size];
1463 }

1465 static struct token *handle_qualifiers(struct token *t, struct decl_state *ctx)
1466 {
1467         while (token_type(t) == TOKEN_IDENT) {
1468                 struct symbol *s = lookup_symbol(t->ident, NS_TYPEDEF);
1469                 if (!s)
1470                         break;
1471                 if (s->type != SYM_KEYWORD)
1472                         break;
1473                 if (!(s->op->type & (KW_ATTRIBUTE | KW_QUALIFIER)))
1474                         break;
1475                 t = t->next;
1476                 if (s->op->declarator)
1477                         t = s->op->declarator(t, ctx);
1478         }
1479         return t;
1480 }

1482 static struct token *declaration_specifiers(struct token *token, struct decl_sta
1483 {
1484         int seen = 0;
1485         int class = CInt;
1486         int size = 0;

1488         while (token_type(token) == TOKEN_IDENT) {
1489                 struct symbol *s = lookup_symbol(token->ident,
1490                                                  NS_TYPEDEF | NS_SYMBOL);
1491                 if (!s || !(s->namespace & NS_TYPEDEF))
1492                         break;
1493                 if (s->type != SYM_KEYWORD) {
1494                         if (seen & Set_Any)
1495                                 break;
1496                         seen |= Set_S | Set_T;
1497                         ctx->ctype.base_type = s->ctype.base_type;
1498                         apply_ctype(token->pos, &s->ctype, &ctx->ctype);
1499                         token = token->next;
1500                         continue;
1501                 }
1502                 if (s->op->type & KW_SPECIFIER) {
1503                         if (seen & s->op->test) {
1504                                 specifier_conflict(token->pos,
1505                                                    seen & s->op->test,
1506                                                    token->ident);
1507                                 break;
1508                         }
1509                         seen |= s->op->set;
1510                         class += s->op->class;
1511                         if (s->op->set & Set_Int128)
1512                                 size = 2;
```



```
1513                         if (s->op->type & KW_SHORT) {
1514                                 size = -1;
1515                         } else if (s->op->type & KW_LONG && size++) {
1516                                 if (class == CReal) {
1517                                         specifier_conflict(token->pos,
1518                                                            Set_Vlong,
1519                                                            &double_ident);
1520                                         break;
1521                                 }
1522                                 seen |= Set_Vlong;
1523                         }
1524                 }
1525                 token = token->next;
1526                 if (s->op->declarator)
1527                         token = s->op->declarator(token, ctx);
1528                 if (s->op->type & KW_EXACT) {
1529                         ctx->ctype.base_type = s->ctype.base_type;
1530                         ctx->ctype.modifiers |= s->ctype.modifiers;
1531                 }
1532         }

1534         if (!(seen & Set_S)) {  /* not set explicitly? */
1535                 struct symbol *base = &incomplete_ctype;
1536                 if (seen & Set_Any)
1537                         base = types[class][size];
1538                 ctx->ctype.base_type = base;
1539         }

1541         if (ctx->ctype.modifiers & MOD_BITWISE) {
1542                 struct symbol *type;
1543                 ctx->ctype.modifiers &= ~MOD_BITWISE;
1544                 if (!is_int_type(ctx->ctype.base_type)) {
1545                         sparse_error(token->pos, "invalid modifier");
1546                         return token;
1547                 }
1548                 type = alloc_symbol(token->pos, SYM_BASETYPE);
1549                 *type = *ctx->ctype.base_type;
1550                 type->ctype.modifiers &= ~MOD_SPECIFIER;
1551                 type->ctype.base_type = ctx->ctype.base_type;
1552                 type->type = SYM_RESTRICT;
1553                 ctx->ctype.base_type = type;
1554                 create_fouled(type);
1555         }
1556         return token;
1557 }

1559 static struct token *abstract_array_static_declarator(struct token *token, int *
1560 {
1561         while (token->ident == &static_ident) {
1562                 if (*has_static)
1563                         sparse_error(token->pos, "duplicate array static declara

1565                 *has_static = 1;
1566                 token = token->next;
1567         }
1568         return token;

1570 }

1572 static struct token *abstract_array_declarator(struct token *token, struct symbo
1573 {
1574         struct expression *expr = NULL;
1575         int has_static = 0;

1577         token = abstract_array_static_declarator(token, &has_static);
```

```
1579         if (match_idents(token, &restrict_ident, &__restrict_ident, &__restrict_
1580                 token = abstract_array_static_declarator(token->next, &has_stati
1581         token = parse_expression(token, &expr);
1582         sym->array_size = expr;
1583         return token;
1584 }

1586 static struct token *parameter_type_list(struct token *, struct symbol *);
1587 static struct token *identifier_list(struct token *, struct symbol *);
1588 static struct token *declarator(struct token *token, struct decl_state *ctx);

1590 static struct token *skip_attribute(struct token *token)
1591 {
1592         token = token->next;
1593         if (match_op(token, '(')) {
1594                 int depth = 1;
1595                 token = token->next;
1596                 while (depth && !eof_token(token)) {
1597                         if (token_type(token) == TOKEN_SPECIAL) {
1598                                 if (token->special == '(')
1599                                         depth++;
1600                                 else if (token->special == ')')
1601                                         depth--;
1602                         }
1603                         token = token->next;
1604                 }
1605         }
1606         return token;
1607 }

1609 static struct token *skip_attributes(struct token *token)
1610 {
1611         struct symbol *keyword;
1612         for (;;) {
1613                 if (token_type(token) != TOKEN_IDENT)
1614                         break;
1615                 keyword = lookup_keyword(token->ident, NS_KEYWORD | NS_TYPEDEF);
1616                 if (!keyword || keyword->type != SYM_KEYWORD)
1617                         break;
1618                 if (!(keyword->op->type & KW_ATTRIBUTE))
1619                         break;
1620                 token = expect(token->next, '(', "after attribute");
1621                 token = expect(token, '(', "after attribute");
1622                 for (;;) {
1623                         if (eof_token(token))
1624                                 break;
1625                         if (match_op(token, ';'))
1626                                 break;
1627                         if (token_type(token) != TOKEN_IDENT)
1628                                 break;
1629                         token = skip_attribute(token);
1630                         if (!match_op(token, ','))
1631                                 break;
1632                         token = token->next;
1633                 }
1634                 token = expect(token, ')', "after attribute");
1635                 token = expect(token, ')', "after attribute");
1636         }
1637         return token;
1638 }

1640 static struct token *handle_attributes(struct token *token, struct decl_state *c
1641 {
1642         struct symbol *keyword;
1643         for (;;) {
1644                 if (token_type(token) != TOKEN_IDENT)
```



```
1645                         break;
1646                 keyword = lookup_keyword(token->ident, NS_KEYWORD | NS_TYPEDEF);
1647                 if (!keyword || keyword->type != SYM_KEYWORD)
1648                         break;
1649                 if (!(keyword->op->type & keywords))
1650                         break;
1651                 token = keyword->op->declarator(token->next, ctx);
1652                 keywords &= KW_ATTRIBUTE;
1653         }
1654         return token;
1655 }

1657 static int is_nested(struct token *token, struct token **p,
1658                      int prefer_abstract)
1659 {
1660         /*
1661          * This can be either a parameter list or a grouping.
1662          * For the direct (non-abstract) case, we know if must be
1663          * a parameter list if we already saw the identifier.
1664          * For the abstract case, we know if must be a parameter
1665          * list if it is empty or starts with a type.
1666          */
1667         struct token *next = token->next;

1669         *p = next = skip_attributes(next);

1671         if (token_type(next) == TOKEN_IDENT) {
1672                 if (lookup_type(next))
1673                         return !prefer_abstract;
1674                 return 1;
1675         }

1677         if (match_op(next, ')') || match_op(next, SPECIAL_ELLIPSIS))
1678                 return 0;

1680         return 1;
1681 }

1683 enum kind {
1684         Empty, K_R, Proto, Bad_Func,
1685 };

1687 static enum kind which_func(struct token *token,
1688                             struct ident **n,
1689                             int prefer_abstract)
1690 {
1691         struct token *next = token->next;

1693         if (token_type(next) == TOKEN_IDENT) {
1694                 if (lookup_type(next))
1695                         return Proto;
1696                 /* identifier list not in definition; complain */
1697                 if (prefer_abstract)
1698                         warning(token->pos,
1699                                 "identifier list not in definition");
1700                 return K_R;
1701         }

1703         if (token_type(next) != TOKEN_SPECIAL)
1704                 return Bad_Func;

1706         if (next->special == ')') {
1707                 /* don't complain about those */
1708                 if (!n || match_op(next->next, ';'))
1709                 if (!n || match_op(next->next, ';') || match_op(next->next, ',')
1710                         return Empty;
```

```
1711                 if (Wstrict_prototypes)
1712                         warning(next->pos,
1713                                 "non-ANSI function declaration of function '%s'"
1714                                 show_ident(*n));
1715                 return Empty;
1716         }

1718         if (next->special == SPECIAL_ELLIPSIS) {
1719                 warning(next->pos,
1720                         "variadic functions must have one named argument");
1721                 return Proto;
1722         }

1724         return Bad_Func;
1725 }

1727 static struct token *direct_declarator(struct token *token, struct decl_state *c
1728 {
1729         struct ctype *ctype = &ctx->ctype;
1730         struct token *next;
1731         struct ident **p = ctx->ident;

1733         if (ctx->ident && token_type(token) == TOKEN_IDENT) {
1734                 *ctx->ident = token->ident;
1735                 token = token->next;
1736         } else if (match_op(token, '(') &&
1737             is_nested(token, &next, ctx->prefer_abstract)) {
1738                 struct symbol *base_type = ctype->base_type;
1739                 if (token->next != next)
1740                         next = handle_attributes(token->next, ctx,
1741                                                  KW_ATTRIBUTE);
1742                 token = declarator(next, ctx);
1743                 token = expect(token, ')', "in nested declarator");
1744                 while (ctype->base_type != base_type)
1745                         ctype = &ctype->base_type->ctype;
1746                 p = NULL;
1747         }

1749         if (match_op(token, '(')) {
1750                 enum kind kind = which_func(token, p, ctx->prefer_abstract);
1751                 struct symbol *fn;
1752                 fn = alloc_indirect_symbol(token->pos, ctype, SYM_FN);
1753                 token = token->next;
1754                 if (kind == K_R)
1755                         token = identifier_list(token, fn);
1756                 else if (kind == Proto)
1757                         token = parameter_type_list(token, fn);
1758                 token = expect(token, ')', "in function declarator");
1759                 fn->endpos = token->pos;
1760                 return token;
1761         }

1763         while (match_op(token, '[')) {
1764                 struct symbol *array;
1765                 array = alloc_indirect_symbol(token->pos, ctype, SYM_ARRAY);
1766                 token = abstract_array_declarator(token->next, array);
1767                 token = expect(token, ']', "in abstract_array_declarator");
1768                 array->endpos = token->pos;
1769                 ctype = &array->ctype;
1770         }
1771         return token;
1772 }

1774 static struct token *pointer(struct token *token, struct decl_state *ctx)
1775 {
1776         while (match_op(token,'*')) {
```



```
1777                 struct symbol *ptr = alloc_symbol(token->pos, SYM_PTR);
1778                 ptr->ctype.modifiers = ctx->ctype.modifiers;
1779                 ptr->ctype.base_type = ctx->ctype.base_type;
1780                 ptr->ctype.as = ctx->ctype.as;
1781                 ptr->ctype.contexts = ctx->ctype.contexts;
1782                 ctx->ctype.modifiers = 0;
1783                 ctx->ctype.base_type = ptr;
1784                 ctx->ctype.as = 0;
1785                 ctx->ctype.contexts = NULL;
1786                 ctx->ctype.alignment = 0;

1788                 token = handle_qualifiers(token->next, ctx);
1789                 ctx->ctype.base_type->endpos = token->pos;
1790         }
1791         return token;
1792 }

1794 static struct token *declarator(struct token *token, struct decl_state *ctx)
1795 {
1796         token = pointer(token, ctx);
1797         return direct_declarator(token, ctx);
1798 }

1800 static struct token *handle_bitfield(struct token *token, struct decl_state *ctx
1801 {
1802         struct ctype *ctype = &ctx->ctype;
1803         struct expression *expr;
1804         struct symbol *bitfield;
1805         long long width;

1807         if (ctype->base_type != &int_type && !is_int_type(ctype->base_type)) {
1808                 sparse_error(token->pos, "invalid bitfield specifier for type %s
1809                         show_typename(ctype->base_type));
1810                 // Parse this to recover gracefully.
1811                 return conditional_expression(token->next, &expr);
1812         }

1814         bitfield = alloc_indirect_symbol(token->pos, ctype, SYM_BITFIELD);
1815         token = conditional_expression(token->next, &expr);
1816         width = const_expression_value(expr);
1817         bitfield->bit_size = width;

1819         if (width < 0 || width > INT_MAX) {
1820                 sparse_error(token->pos, "invalid bitfield width, %lld.", width)
1821                 width = -1;
1822         } else if (*ctx->ident && width == 0) {
1823                 sparse_error(token->pos, "invalid named zero-width bitfield '%s'
1824                         show_ident(*ctx->ident));
1825                 width = -1;
1826         } else if (*ctx->ident) {
1827                 struct symbol *base_type = bitfield->ctype.base_type;
1828                 struct symbol *bitfield_type = base_type == &int_type ? bitfield
1829                 int is_signed = !(bitfield_type->ctype.modifiers & MOD_UNSIGNED)
1830                 if (Wone_bit_signed_bitfield && width == 1 && is_signed) {
1831                         // Valid values are either {-1;0} or {0}, depending on i
1832                         // representation.  The latter makes for very efficient
1833                         sparse_error(token->pos, "dubious one-bit signed bitfiel
1834                 }
1835                 if (Wdefault_bitfield_sign &&
1836                     bitfield_type->type != SYM_ENUM &&
1837                     !(bitfield_type->ctype.modifiers & MOD_EXPLICITLY_SIGNED) &&
1838                     is_signed) {
1839                         // The sign of bitfields is unspecified by default.
1840                         warning(token->pos, "dubious bitfield without explicit '
1841                 }
1842         }
```

```
1843         bitfield->bit_size = width;
1844         bitfield->endpos = token->pos;
1845         return token;
1846 }

1848 static struct token *declaration_list(struct token *token, struct symbol_list **
1849 {
1850         struct decl_state ctx = {.prefer_abstract = 0};
1851         struct ctype saved;
1852         unsigned long mod;

1854         token = declaration_specifiers(token, &ctx);
1855         mod = storage_modifiers(&ctx);
1856         saved = ctx.ctype;
1857         for (;;) {
1858                 struct symbol *decl = alloc_symbol(token->pos, SYM_NODE);
1859                 ctx.ident = &decl->ident;

1861                 token = declarator(token, &ctx);
1862                 if (match_op(token, ':'))
1863                         token = handle_bitfield(token, &ctx);

1865                 token = handle_attributes(token, &ctx, KW_ATTRIBUTE);
1866                 apply_modifiers(token->pos, &ctx);

1868                 decl->ctype = ctx.ctype;
1869                 decl->ctype.modifiers |= mod;
1870                 decl->endpos = token->pos;
1871                 add_symbol(list, decl);
1872                 if (!match_op(token, ','))
1873                         break;
1874                 token = token->next;
1875                 ctx.ctype = saved;
1876         }
1877         return token;
1878 }

1880 static struct token *struct_declaration_list(struct token *token, struct symbol_
1881 {
1882         while (!match_op(token, '}')) {
1883                 if (match_ident(token, &_Static_assert_ident)) {
1884                         token = parse_static_assert(token, NULL);
1885                         continue;
1886                 }
1887                 if (!match_op(token, ';'))
1888                         token = declaration_list(token, list);
1889                 if (!match_op(token, ';')) {
1890                         sparse_error(token->pos, "expected ; at end of declarati
1891                         break;
1892                 }
1893                 token = token->next;
1894         }
1895         return token;
1896 }

1898 static struct token *parameter_declaration(struct token *token, struct symbol *s
1899 {
1900         struct decl_state ctx = {.prefer_abstract = 1};

1902         token = declaration_specifiers(token, &ctx);
1903         ctx.ident = &sym->ident;
1904         token = declarator(token, &ctx);
1905         token = handle_attributes(token, &ctx, KW_ATTRIBUTE);
1906         apply_modifiers(token->pos, &ctx);
1907         sym->ctype = ctx.ctype;
1908         sym->ctype.modifiers |= storage_modifiers(&ctx);
```



```
1909         sym->endpos = token->pos;
1910         sym->forced_arg = ctx.storage_class == SForced;
1911         return token;
1912 }

1914 struct token *typename(struct token *token, struct symbol **p, int *forced)
1915 {
1916         struct decl_state ctx = {.prefer_abstract = 1};
1917         int class;
1918         struct symbol *sym = alloc_symbol(token->pos, SYM_NODE);
1919         *p = sym;
1920         token = declaration_specifiers(token, &ctx);
1921         token = declarator(token, &ctx);
1922         apply_modifiers(token->pos, &ctx);
1923         sym->ctype = ctx.ctype;
1924         sym->endpos = token->pos;
1925         class = ctx.storage_class;
1926         if (forced) {
1927                 *forced = 0;
1928                 if (class == SForced) {
1929                         *forced = 1;
1930                         class = 0;
1931                 }
1932         }
1933         if (class)
1934                 warning(sym->pos, "storage class in typename (%s %s)",
1935                         storage_class[class], show_typename(sym));
1936         return token;
1937 }

1939 static struct token *parse_underscore_Pragma(struct token *token)
1940 {
1941         struct token *next;

1943         next = token->next;
1944         if (!match_op(next, '('))
1945                 return next;
1946         next = next->next;
1947         if (next->pos.type != TOKEN_STRING)
1948                 return next;
1949         next = next->next;
1950         if (!match_op(next, ')'))
1951                 return next;
1952         return next->next;
1953 }

1955 static struct token *expression_statement(struct token *token, struct expression
1956 {
1957         if (match_ident(token, &_Pragma_ident))
1958                 return parse_underscore_Pragma(token);

1960         token = parse_expression(token, tree);
1961         return expect(token, ';', "at end of statement");
1962 }

1964 static struct token *parse_asm_operands(struct token *token, struct statement *s
1965         struct expression_list **inout)
1966 {
1967         struct expression *expr;

1969         /* Allow empty operands */
1970         if (match_op(token->next, ':') || match_op(token->next, ')'))
1971                 return token->next;
1972         do {
1973                 struct ident *ident = NULL;
1974                 if (match_op(token->next, '[') &&
```

```
1975                    token_type(token->next->next) == TOKEN_IDENT &&
1976                    match_op(token->next->next->next, ']')) {
1977                        ident = token->next->next->ident;
1978                        token = token->next->next->next;
1979                }
1980                add_expression(inout, (struct expression *)ident); /* UGGLEE!!!
1981                token = primary_expression(token->next, &expr);
1982                add_expression(inout, expr);
1983                token = parens_expression(token, &expr, "in asm parameter");
1984                add_expression(inout, expr);
1985        } while (match_op(token, ','));
1986        return token;
1987 }

1989 static struct token *parse_asm_clobbers(struct token *token, struct statement *s
1990        struct expression_list **clobbers)
1991 {
1992        struct expression *expr;

1994        do {
1995                token = primary_expression(token->next, &expr);
1996                if (expr)
1997                        add_expression(clobbers, expr);
1998        } while (match_op(token, ','));
1999        return token;
2000 }

2002 static struct token *parse_asm_labels(struct token *token, struct statement *stm
2003                        struct symbol_list **labels)
2004 {
2005        struct symbol *label;

2007        do {
2008                token = token->next; /* skip ':' and ',' */
2009                if (token_type(token) != TOKEN_IDENT)
2010                        return token;
2011                label = label_symbol(token);
2012                add_symbol(labels, label);
2013                token = token->next;
2014        } while (match_op(token, ','));
2015        return token;
2016 }

2018 static struct token *parse_asm_statement(struct token *token, struct statement *
2019 {
2020        int is_goto = 0;

2022        token = token->next;
2023        stmt->type = STMT_ASM;
2024        if (match_idents(token, &__volatile___ident, &__volatile_ident, &volatil
2025                token = token->next;
2026        }
2027        if (token_type(token) == TOKEN_IDENT && token->ident == &goto_ident) {
2028                is_goto = 1;
2029                token = token->next;
2030        }
2031        token = expect(token, '(', "after asm");
2032        token = parse_expression(token, &stmt->asm_string);
2033        if (match_op(token, ':'))
2034                token = parse_asm_operands(token, stmt, &stmt->asm_outputs);
2035        if (match_op(token, ':'))
2036                token = parse_asm_operands(token, stmt, &stmt->asm_inputs);
2037        if (match_op(token, ':'))
2038                token = parse_asm_clobbers(token, stmt, &stmt->asm_clobbers);
2039        if (is_goto && match_op(token, ':'))
2040                token = parse_asm_labels(token, stmt, &stmt->asm_labels);
```



```
2041        token = expect(token, ')', "after asm");
2042        return expect(token, ';', "at end of asm-statement");
2043 }

2045 static struct token *parse_asm_declarator(struct token *token, struct decl_state
2046 {
2047        struct expression *expr;
2048        token = expect(token, '(', "after asm");
2049        token = parse_expression(token->next, &expr);
2050        token = expect(token, ')', "after asm");
2051        return token;
2052 }

2054 static struct token *parse_static_assert(struct token *token, struct symbol_list
2055 {
2056        struct expression *cond = NULL, *message = NULL;

2058        token = expect(token->next, '(', "after _Static_assert");
2059        token = constant_expression(token, &cond);
2060        if (!cond)
2061                sparse_error(token->pos, "Expected constant expression");
2062        token = expect(token, ',', "after conditional expression in _Static_asse
2063        token = parse_expression(token, &message);
2064        if (!message || message->type != EXPR_STRING) {
2065                struct position pos;

2067                pos = message ? message->pos : token->pos;
2068                sparse_error(pos, "bad or missing string literal");
2069                cond = NULL;
2070        }
2071        token = expect(token, ')', "after diagnostic message in _Static_assert")

2073        token = expect(token, ';', "after _Static_assert()");

2075        if (cond && !const_expression_value(cond) && cond->type == EXPR_VALUE)
2076                sparse_error(cond->pos, "static assertion failed: %s",
2077                                show_string(message->string));
2078        return token;
2079 }

2081 /* Make a statement out of an expression */
2082 static struct statement *make_statement(struct expression *expr)
2083 {
2084        struct statement *stmt;

2086        if (!expr)
2087                return NULL;
2088        stmt = alloc_statement(expr->pos, STMT_EXPRESSION);
2089        stmt->expression = expr;
2090        return stmt;
2091 }

2093 /*
2094  * All iterators have two symbols associated with them:
2095  * the "continue" and "break" symbols, which are targets
2096  * for continue and break statements respectively.
2097  *
2098  * They are in a special name-space, but they follow
2099  * all the normal visibility rules, so nested iterators
2100  * automatically work right.
2101  */
2102 static void start_iterator(struct statement *stmt)
2103 {
2104        struct symbol *cont, *brk;

2106        start_symbol_scope(stmt->pos);
```

```
2107         cont = alloc_symbol(stmt->pos, SYM_NODE);
2108         bind_symbol(cont, &continue_ident, NS_ITERATOR);
2109         brk = alloc_symbol(stmt->pos, SYM_NODE);
2110         bind_symbol(brk, &break_ident, NS_ITERATOR);

2112         stmt->type = STMT_ITERATOR;
2113         stmt->iterator_break = brk;
2114         stmt->iterator_continue = cont;
2115         fn_local_symbol(brk);
2116         fn_local_symbol(cont);
2117 }

2119 static void end_iterator(struct statement *stmt)
2120 {
2121         end_symbol_scope();
2122 }

2124 static struct statement *start_function(struct symbol *sym)
2125 {
2126         struct symbol *ret;
2127         struct statement *stmt = alloc_statement(sym->pos, STMT_COMPOUND);

2129         start_function_scope(sym->pos);
2130         ret = alloc_symbol(sym->pos, SYM_NODE);
2131         ret->ctype = sym->ctype.base_type->ctype;
2132         ret->ctype.modifiers &= ~(MOD_STORAGE | MOD_CONST | MOD_VOLATILE | MOD_T
2133         ret->ctype.modifiers |= (MOD_AUTO | MOD_REGISTER);
2134         bind_symbol(ret, &return_ident, NS_ITERATOR);
2135         stmt->ret = ret;
2136         fn_local_symbol(ret);

2138         // Currently parsed symbol for __func__/__FUNCTION__/__PRETTY_FUNCTION__
2139         current_fn = sym;

2141         return stmt;
2142 }

2144 static void end_function(struct symbol *sym)
2145 {
2146         current_fn = NULL;
2147         end_function_scope();
2148 }

2150 /*
2151  * A "switch()" statement, like an iterator, has a
2152  * the "break" symbol associated with it. It works
2153  * exactly like the iterator break - it's the target
2154  * for any break-statements in scope, and means that
2155  * "break" handling doesn't even need to know whether
2156  * it's breaking out of an iterator or a switch.
2157  *
2158  * In addition, the "case" symbol is a marker for the
2159  * case/default statements to find the switch statement
2160  * that they are associated with.
2161  */
2162 static void start_switch(struct statement *stmt)
2163 {
2164         struct symbol *brk, *switch_case;

2166         start_symbol_scope(stmt->pos);
2167         brk = alloc_symbol(stmt->pos, SYM_NODE);
2168         bind_symbol(brk, &break_ident, NS_ITERATOR);

2170         switch_case = alloc_symbol(stmt->pos, SYM_NODE);
2171         bind_symbol(switch_case, &case_ident, NS_ITERATOR);
2172         switch_case->stmt = stmt;
```



```
2174         stmt->type = STMT_SWITCH;
2175         stmt->switch_break = brk;
2176         stmt->switch_case = switch_case;

2178         fn_local_symbol(brk);
2179         fn_local_symbol(switch_case);
2180 }

2182 static void end_switch(struct statement *stmt)
2183 {
2184         if (!stmt->switch_case->symbol_list)
2185                 warning(stmt->pos, "switch with no cases");
2186         end_symbol_scope();
2187 }

2189 static void add_case_statement(struct statement *stmt)
2190 {
2191         struct symbol *target = lookup_symbol(&case_ident, NS_ITERATOR);
2192         struct symbol *sym;

2194         if (!target) {
2195                 sparse_error(stmt->pos, "not in switch scope");
2196                 stmt->type = STMT_NONE;
2197                 return;
2198         }
2199         sym = alloc_symbol(stmt->pos, SYM_NODE);
2200         add_symbol(&target->symbol_list, sym);
2201         sym->stmt = stmt;
2202         stmt->case_label = sym;
2203         fn_local_symbol(sym);
2204 }

2206 static struct token *parse_return_statement(struct token *token, struct statemen
2207 {
2208         struct symbol *target = lookup_symbol(&return_ident, NS_ITERATOR);

2210         if (!target)
2211                 error_die(token->pos, "internal error: return without a function
2212         stmt->type = STMT_RETURN;
2213         stmt->ret_target = target;
2214         return expression_statement(token->next, &stmt->ret_value);
2215 }

2217 static void validate_for_loop_decl(struct symbol *sym)
2218 {
2219         unsigned long storage = sym->ctype.modifiers & MOD_STORAGE;

2221         if (storage & ~(MOD_AUTO | MOD_REGISTER)) {
2222                 const char *name = show_ident(sym->ident);
2223                 sparse_error(sym->pos, "non-local var '%s' in for-loop initializ
2224                 sym->ctype.modifiers &= ~MOD_STORAGE;
2225         }
2226 }

2228 static struct token *parse_for_statement(struct token *token, struct statement *
2229 {
2230         struct symbol_list *syms;
2231         struct expression *e1, *e2, *e3;
2232         struct statement *iterator;

2234         start_iterator(stmt);
2235         token = expect(token->next, '(', "after 'for'");

2237         syms = NULL;
2238         e1 = NULL;
```

```
2239         /* C99 variable declaration? */
2240         if (lookup_type(token)) {
2241                 token = external_declaration(token, &syms, validate_for_loop_dec
2242         } else {
2243                 token = parse_expression(token, &e1);
2244                 token = expect(token, ';', "in 'for'");
2245         }
2246         token = parse_expression(token, &e2);
2247         token = expect(token, ';', "in 'for'");
2248         token = parse_expression(token, &e3);
2249         token = expect(token, ')', "in 'for'");
2250         token = statement(token, &iterator);

2252         stmt->iterator_syms = syms;
2253         stmt->iterator_pre_statement = make_statement(e1);
2254         stmt->iterator_pre_condition = e2;
2255         stmt->iterator_post_statement = make_statement(e3);
2256         stmt->iterator_post_condition = NULL;
2257         stmt->iterator_statement = iterator;
2258         end_iterator(stmt);

2260         return token;
2261 }

2263 static struct token *parse_while_statement(struct token *token, struct statement
2264 {
2265         struct expression *expr;
2266         struct statement *iterator;

2268         start_iterator(stmt);
2269         token = parens_expression(token->next, &expr, "after 'while'");
2270         token = statement(token, &iterator);

2272         stmt->iterator_pre_condition = expr;
2273         stmt->iterator_post_condition = NULL;
2274         stmt->iterator_statement = iterator;
2275         end_iterator(stmt);

2277         return token;
2278 }

2280 static struct token *parse_do_statement(struct token *token, struct statement *s
2281 {
2282         struct expression *expr;
2283         struct statement *iterator;

2285         start_iterator(stmt);
2286         token = statement(token->next, &iterator);
2287         if (token_type(token) == TOKEN_IDENT && token->ident == &while_ident)
2288                 token = token->next;
2289         else
2290                 sparse_error(token->pos, "expected 'while' after 'do'");
2291         token = parens_expression(token, &expr, "after 'do-while'");

2293         stmt->iterator_post_condition = expr;
2294         stmt->iterator_statement = iterator;
2295         end_iterator(stmt);

2297         if (iterator && iterator->type != STMT_COMPOUND && Wdo_while)
2298                 warning(iterator->pos, "do-while statement is not a compound sta

2300         return expect(token, ';', "after statement");
2301 }

2303 static struct token *parse_if_statement(struct token *token, struct statement *s
2304 {
```



```
2305         stmt->type = STMT_IF;
2306         token = parens_expression(token->next, &stmt->if_conditional, "after if"
2307         token = statement(token, &stmt->if_true);
2308         if (token_type(token) != TOKEN_IDENT)
2309                 return token;
2310         if (token->ident != &else_ident)
2311                 return token;
2312         return statement(token->next, &stmt->if_false);
2313 }

2315 static inline struct token *case_statement(struct token *token, struct statement
2316 {
2317         stmt->type = STMT_CASE;
2318         token = expect(token, ':', "after default/case");
2319         add_case_statement(stmt);
2320         return statement(token, &stmt->case_statement);
2321 }

2323 static struct token *parse_case_statement(struct token *token, struct statement
2324 {
2325         token = parse_expression(token->next, &stmt->case_expression);
2326         if (match_op(token, SPECIAL_ELLIPSIS))
2327                 token = parse_expression(token->next, &stmt->case_to);
2328         return case_statement(token, stmt);
2329 }

2331 static struct token *parse_default_statement(struct token *token, struct stateme
2332 {
2333         return case_statement(token->next, stmt);
2334 }

2336 static struct token *parse_loop_iterator(struct token *token, struct statement *
2337 {
2338         struct symbol *target = lookup_symbol(token->ident, NS_ITERATOR);
2339         stmt->type = STMT_GOTO;
2340         stmt->goto_label = target;
2341         if (!target)
2342                 sparse_error(stmt->pos, "break/continue not in iterator scope");
2343         return expect(token->next, ';', "at end of statement");
2344 }

2346 static struct token *parse_switch_statement(struct token *token, struct statemen
2347 {
2348         stmt->type = STMT_SWITCH;
2349         start_switch(stmt);
2350         token = parens_expression(token->next, &stmt->switch_expression, "after
2351         token = statement(token, &stmt->switch_statement);
2352         end_switch(stmt);
2353         return token;
2354 }

2356 static struct token *parse_goto_statement(struct token *token, struct statement
2357 {
2358         stmt->type = STMT_GOTO;
2359         token = token->next;
2360         if (match_op(token, '*')) {
2361                 token = parse_expression(token->next, &stmt->goto_expression);
2362                 add_statement(&function_computed_goto_list, stmt);
2363         } else if (token_type(token) == TOKEN_IDENT) {
2364                 stmt->goto_label = label_symbol(token);
2365                 token = token->next;
2366         } else {
2367                 sparse_error(token->pos, "Expected identifier or goto expression
2368         }
2369         return expect(token, ';', "at end of statement");
2370 }
```

```
2372 static struct token *parse_context_statement(struct token *token, struct stateme
2373 {
2374         stmt->type = STMT_CONTEXT;
2375         token = parse_expression(token->next, &stmt->expression);
2376         if (stmt->expression->type == EXPR_PREOP
2377             && stmt->expression->op == '('
2378             && stmt->expression->unop->type == EXPR_COMMA) {
2379                 struct expression *expr;
2380                 expr = stmt->expression->unop;
2381                 stmt->context = expr->left;
2382                 stmt->expression = expr->right;
2383         }
2384         return expect(token, ';', "at end of statement");
2385 }

2387 static struct token *parse_range_statement(struct token *token, struct statement
2388 {
2389         stmt->type = STMT_RANGE;
2390         token = assignment_expression(token->next, &stmt->range_expression);
2391         token = expect(token, ',', "after range expression");
2392         token = assignment_expression(token, &stmt->range_low);
2393         token = expect(token, ',', "after low range");
2394         token = assignment_expression(token, &stmt->range_high);
2395         return expect(token, ';', "after range statement");
2396 }

2398 static struct token *statement(struct token *token, struct statement **tree)
2399 {
2400         struct statement *stmt = alloc_statement(token->pos, STMT_NONE);

2402         *tree = stmt;
2403         if (token_type(token) == TOKEN_IDENT) {
2404                 struct symbol *s = lookup_keyword(token->ident, NS_KEYWORD);
2405                 if (s && s->op->statement)
2406                         return s->op->statement(token, stmt);

2408                 if (match_op(token->next, ':')) {
2409                         struct symbol *s = label_symbol(token);
2410                         stmt->type = STMT_LABEL;
2411                         stmt->label_identifier = s;
2412                         if (s->stmt)
2413                                 sparse_error(stmt->pos, "label '%s' redefined",
2414                         s->stmt = stmt;
2415                         token = skip_attributes(token->next->next);
2416                         return statement(token, &stmt->label_statement);
2417                 }
2418         }

2420         if (match_op(token, '{')) {
2421                 stmt->type = STMT_COMPOUND;
2422                 start_symbol_scope(stmt->pos);
2423                 token = compound_statement(token->next, stmt);
2424                 end_symbol_scope();
2425
2426                 return expect(token, '}', "at end of compound statement");
2427         }
2428
2429         stmt->type = STMT_EXPRESSION;
2430         return expression_statement(token, &stmt->expression);
2431 }

2433 /* gcc extension - __label__ ident-list; in the beginning of compound stmt */
2434 static struct token *label_statement(struct token *token)
2435 {
2436         while (token_type(token) == TOKEN_IDENT) {
```



```
2437                 struct symbol *sym = alloc_symbol(token->pos, SYM_LABEL);
2438                 /* it's block-scope, but we want label namespace */
2439                 bind_symbol(sym, token->ident, NS_SYMBOL);
2440                 sym->namespace = NS_LABEL;
2441                 fn_local_symbol(sym);
2442                 token = token->next;
2443                 if (!match_op(token, ','))
2444                         break;
2445                 token = token->next;
2446         }
2447         return expect(token, ';', "at end of label declaration");
2448 }

2450 static struct token * statement_list(struct token *token, struct statement_list
2451 {
2452         int seen_statement = 0;
2453         while (token_type(token) == TOKEN_IDENT &&
2454                token->ident == &__label___ident)
2455                 token = label_statement(token->next);
2456         for (;;) {
2457                 struct statement * stmt;
2458                 if (eof_token(token))
2459                         break;
2460                 if (match_op(token, '}'))
2461                         break;
2462                 if (match_ident(token, &_Static_assert_ident)) {
2463                         token = parse_static_assert(token, NULL);
2464                         continue;
2465                 }
2466                 if (lookup_type(token)) {
2467                         if (seen_statement) {
2468                                 warning(token->pos, "mixing declarations and cod
2469                                 seen_statement = 0;
2470                         }
2471                         stmt = alloc_statement(token->pos, STMT_DECLARATION);
2472                         token = external_declaration(token, &stmt->declaration,
2473                 } else {
2474                         seen_statement = Wdeclarationafterstatement;
2475                         token = statement(token, &stmt);
2476                 }
2477                 add_statement(list, stmt);
2478         }
2479         return token;
2480 }

2482 static struct token *identifier_list(struct token *token, struct symbol *fn)
2483 {
2484         struct symbol_list **list = &fn->arguments;
2485         for (;;) {
2486                 struct symbol *sym = alloc_symbol(token->pos, SYM_NODE);
2487                 sym->ident = token->ident;
2488                 token = token->next;
2489                 sym->endpos = token->pos;
2490                 sym->ctype.base_type = &incomplete_ctype;
2491                 add_symbol(list, sym);
2492                 if (!match_op(token, ',') ||
2493                     token_type(token->next) != TOKEN_IDENT ||
2494                     lookup_type(token->next))
2495                         break;
2496                 token = token->next;
2497         }
2498         return token;
2499 }

2501 static struct token *parameter_type_list(struct token *token, struct symbol *fn)
2502 {
```

```
2503         struct symbol_list **list = &fn->arguments;

2505         for (;;) {
2506                 struct symbol *sym;

2508                 if (match_op(token, SPECIAL_ELLIPSIS)) {
2509                         fn->variadic = 1;
2510                         token = token->next;
2511                         break;
2512                 }

2514                 sym = alloc_symbol(token->pos, SYM_NODE);
2515                 token = parameter_declaration(token, sym);
2516                 if (sym->ctype.base_type == &void_ctype) {
2517                         /* Special case: (void) */
2518                         if (!*list && !sym->ident)
2519                                 break;
2520                         warning(token->pos, "void parameter");
2521                 }
2522                 add_symbol(list, sym);
2523                 if (!match_op(token, ','))
2524                         break;
2525                 token = token->next;
2526         }
2527         return token;
2528 }

2530 struct token *compound_statement(struct token *token, struct statement *stmt)
2531 {
2532         token = statement_list(token, &stmt->stmts);
2533         return token;
2534 }

2536 static struct expression *identifier_expression(struct token *token)
2537 {
2538         struct expression *expr = alloc_expression(token->pos, EXPR_IDENTIFIER);
2539         expr->expr_ident = token->ident;
2540         return expr;
2541 }

2543 static struct expression *index_expression(struct expression *from, struct expre
2544 {
2545         int idx_from, idx_to;
2546         struct expression *expr = alloc_expression(from->pos, EXPR_INDEX);

2548         idx_from = const_expression_value(from);
2549         idx_to = idx_from;
2550         if (to) {
2551                 idx_to = const_expression_value(to);
2552                 if (idx_to < idx_from || idx_from < 0)
2553                         warning(from->pos, "nonsense array initializer index ran
2554         }
2555         expr->idx_from = idx_from;
2556         expr->idx_to = idx_to;
2557         return expr;
2558 }

2560 static struct token *single_initializer(struct expression **ep, struct token *to
2561 {
2562         int expect_equal = 0;
2563         struct token *next = token->next;
2564         struct expression **tail = ep;
2565         int nested;

2567         *ep = NULL;
```



```
2569         if ((token_type(token) == TOKEN_IDENT) && match_op(next, ':')) {
2570                 struct expression *expr = identifier_expression(token);
2571                 if (Wold_initializer)
2572                         warning(token->pos, "obsolete struct initializer, use C9
2573                 token = initializer(&expr->ident_expression, next->next);
2574                 if (expr->ident_expression)
2575                         *ep = expr;
2576                 return token;
2577         }

2579         for (tail = ep, nested = 0; ; nested++, next = token->next) {
2580                 if (match_op(token, '.') && (token_type(next) == TOKEN_IDENT)) {
2581                         struct expression *expr = identifier_expression(next);
2582                         *tail = expr;
2583                         tail = &expr->ident_expression;
2584                         expect_equal = 1;
2585                         token = next->next;
2586                 } else if (match_op(token, '[')) {
2587                         struct expression *from = NULL, *to = NULL, *expr;
2588                         token = constant_expression(token->next, &from);
2589                         if (!from) {
2590                                 sparse_error(token->pos, "Expected constant expr
2591                                 break;
2592                         }
2593                         if (match_op(token, SPECIAL_ELLIPSIS))
2594                                 token = constant_expression(token->next, &to);
2595                         expr = index_expression(from, to);
2596                         *tail = expr;
2597                         tail = &expr->idx_expression;
2598                         token = expect(token, ']', "at end of initializer index"
2599                         if (nested)
2600                                 expect_equal = 1;
2601                 } else {
2602                         break;
2603                 }
2604         }
2605         if (nested && !expect_equal) {
2606                 if (!match_op(token, '='))
2607                         warning(token->pos, "obsolete array initializer, use C99
2608                 else
2609                         expect_equal = 1;
2610         }
2611         if (expect_equal)
2612                 token = expect(token, '=', "at end of initializer index");

2614         token = initializer(tail, token);
2615         if (!*tail)
2616                 *ep = NULL;
2617         return token;
2618 }

2620 static struct token *initializer_list(struct expression_list **list, struct toke
2621 {
2622         struct expression *expr;

2624         for (;;) {
2625                 token = single_initializer(&expr, token);
2626                 if (!expr)
2627                         break;
2628                 add_expression(list, expr);
2629                 if (!match_op(token, ','))
2630                         break;
2631                 token = token->next;
2632         }
2633         return token;
2634 }
```

```
2636 struct token *initializer(struct expression **tree, struct token *token)
2637 {
2638         if (match_op(token, '{')) {
2639                 struct expression *expr = alloc_expression(token->pos, EXPR_INIT
2640                 *tree = expr;
2641                 token = initializer_list(&expr->expr_list, token->next);
2642                 return expect(token, '}', "at end of initializer");
2643         }
2644         return assignment_expression(token, tree);
2645 }

2647 static void declare_argument(struct symbol *sym, struct symbol *fn)
2648 {
2649         if (!sym->ident) {
2650                 sparse_error(sym->pos, "no identifier for function argument");
2651                 return;
2652         }
2653         bind_symbol(sym, sym->ident, NS_SYMBOL);
2654 }

2656 static int is_syscall(struct symbol *sym)
2657 {
2658         char *macro;
2659         char *name;
2660         int is_syscall = 0;

2662         macro = get_macro_name(sym->pos);
2663         if (macro &&
2664             (strncmp("SYSCALL_DEFINE", macro, strlen("SYSCALL_DEFINE")) == 0 ||
2665              strncmp("COMPAT_SYSCALL_DEFINE", macro, strlen("COMPAT_SYSCALL_DEFI
2666                 is_syscall = 1;

2668         name = sym->ident->name;

2670         if (name && strncmp(name, "sys_", 4) ==0)
2671                 is_syscall = 1;
2672
2673         if (name && strncmp(name, "compat_sys_", 11) == 0)
2674                 is_syscall = 1;

2676         return is_syscall;
2677 }
2678


2680 static struct token *parse_function_body(struct token *token, struct symbol *dec
2681         struct symbol_list **list)
2682 {
2683         struct symbol_list **old_symbol_list;
2684         struct symbol *base_type = decl->ctype.base_type;
2685         struct statement *stmt, **p;
2686         struct symbol *prev;
2687         struct symbol *arg;

2689         old_symbol_list = function_symbol_list;
2690         if (decl->ctype.modifiers & MOD_INLINE) {
2691                 function_symbol_list = &decl->inline_symbol_list;
2692                 p = &base_type->inline_stmt;
2693         } else {
2694                 function_symbol_list = &decl->symbol_list;
2695                 p = &base_type->stmt;
2696         }
2697         function_computed_target_list = NULL;
2698         function_computed_goto_list = NULL;

2700         if (decl->ctype.modifiers & MOD_EXTERN &&
```



```
2701                 !(decl->ctype.modifiers & MOD_INLINE) &&
2702                 Wexternal_function_has_definition)
2703                 warning(decl->pos, "function '%s' with external linkage has defi

2705         if (!(decl->ctype.modifiers & MOD_STATIC))
2706                 decl->ctype.modifiers |= MOD_EXTERN;

2708         stmt = start_function(decl);

2710         *p = stmt;
2711         FOR_EACH_PTR (base_type->arguments, arg) {
2712                 declare_argument(arg, base_type);
2713         } END_FOR_EACH_PTR(arg);

2715         token = compound_statement(token->next, stmt);

2717         end_function(decl);
2718         if (!(decl->ctype.modifiers & MOD_INLINE))
2719                 add_symbol(list, decl);
2720         else if (is_syscall(decl)) {
2721             add_symbol(list, decl);
2722             /*
2723             printf("parse.c decl: %s\n", decl->ident->name);
2724             char *macro = get_macro_name(decl->pos);
2725             printf("decl macro: %s\n", macro);
2726             */
2727         }
2728         check_declaration(decl);
2729         decl->definition = decl;
2730         prev = decl->same_symbol;
2731         if (prev && prev->definition) {
2732                 warning(decl->pos, "multiple definitions for function '%s'",
2733                         show_ident(decl->ident));
2734                 info(prev->definition->pos, " the previous one is here");
2735         } else {
2736                 while (prev) {
2737                         rebind_scope(prev, decl->scope);
2738                         prev->definition = decl;
2739                         prev = prev->same_symbol;
2740                 }
2741         }
2742         function_symbol_list = old_symbol_list;
2743         if (function_computed_goto_list) {
2744                 if (!function_computed_target_list)
2745                         warning(decl->pos, "function '%s' has computed goto but
2746                 else {
2747                         FOR_EACH_PTR(function_computed_goto_list, stmt) {
2748                                 stmt->target_list = function_computed_target_lis
2749                         } END_FOR_EACH_PTR(stmt);
2750                 }
2751         }
2752         return expect(token, '}', "at end of function");
2753 }

2755 static void promote_k_r_types(struct symbol *arg)
2756 {
2757         struct symbol *base = arg->ctype.base_type;
2758         if (base && base->ctype.base_type == &int_type && (base->ctype.modifiers
2759                 arg->ctype.base_type = &int_ctype;
2760         }
2761 }

2763 static void apply_k_r_types(struct symbol_list *argtypes, struct symbol *fn)
2764 {
2765         struct symbol_list *real_args = fn->ctype.base_type->arguments;
2766         struct symbol *arg;
```

```
2768         FOR_EACH_PTR(real_args, arg) {
2769                 struct symbol *type;

2771                 /* This is quadratic in the number of arguments. We _really_ don
2772                 FOR_EACH_PTR(argtypes, type) {
2773                         if (type->ident == arg->ident)
2774                                 goto match;
2775                 } END_FOR_EACH_PTR(type);
2776                 if (Wimplicit_int) {
2777                         sparse_error(arg->pos, "missing type declaration for par
2778                                 show_ident(arg->ident));
2779                 }
2780                 continue;
2781 match:
2782                 type->used = 1;
2783                 /* "char" and "short" promote to "int" */
2784                 promote_k_r_types(type);

2786                 arg->ctype = type->ctype;
2787         } END_FOR_EACH_PTR(arg);

2789         FOR_EACH_PTR(argtypes, arg) {
2790                 if (!arg->used)
2791                         warning(arg->pos, "nonsensical parameter declaration '%s
2792         } END_FOR_EACH_PTR(arg);

2794 }

2796 static struct token *parse_k_r_arguments(struct token *token, struct symbol *dec
2797         struct symbol_list **list)
2798 {
2799         struct symbol_list *args = NULL;

2801         if (Wold_style_definition)
2802                 warning(token->pos, "non-ANSI definition of function '%s'", show

2804         do {
2805                 token = declaration_list(token, &args);
2806                 if (!match_op(token, ';')) {
2807                         sparse_error(token->pos, "expected ';' at end of paramet
2808                         break;
2809                 }
2810                 token = token->next;
2811         } while (lookup_type(token));

2813         apply_k_r_types(args, decl);

2815         if (!match_op(token, '{')) {
2816                 sparse_error(token->pos, "expected function body");
2817                 return token;
2818         }
2819         return parse_function_body(token, decl, list);
2820 }

2822 static struct token *toplevel_asm_declaration(struct token *token, struct symbol
2823 {
2824         struct symbol *anon = alloc_symbol(token->pos, SYM_NODE);
2825         struct symbol *fn = alloc_symbol(token->pos, SYM_FN);
2826         struct statement *stmt;

2828         anon->ctype.base_type = fn;
2829         stmt = alloc_statement(token->pos, STMT_NONE);
2830         fn->stmt = stmt;

2832         token = parse_asm_statement(token, stmt);
```



```
2834         add_symbol(list, anon);
2835         return token;
2836 }

2838 struct token *external_declaration(struct token *token, struct symbol_list **lis
2839                 validate_decl_t validate_decl)
2840 {
2841         struct ident *ident = NULL;
2842         struct symbol *decl;
2843         struct decl_state ctx = { .ident = &ident };
2844         struct ctype saved;
2845         struct symbol *base_type;
2846         unsigned long mod;
2847         int is_typedef;

2849         if (match_ident(token, &_Pragma_ident))
2850                 return parse_underscore_Pragma(token);

2852         /* Top-level inline asm or static assertion? */
2853         if (token_type(token) == TOKEN_IDENT) {
2854                 struct symbol *s = lookup_keyword(token->ident, NS_KEYWORD);
2855                 if (s && s->op->toplevel)
2856                         return s->op->toplevel(token, list);
2857         }

2859         /* Parse declaration-specifiers, if any */
2860         token = declaration_specifiers(token, &ctx);
2861         mod = storage_modifiers(&ctx);
2862         decl = alloc_symbol(token->pos, SYM_NODE);
2863         /* Just a type declaration? */
2864         if (match_op(token, ';')) {
2865                 apply_modifiers(token->pos, &ctx);
2866                 return token->next;
2867         }

2869         saved = ctx.ctype;
2870         token = declarator(token, &ctx);
2871         token = handle_attributes(token, &ctx, KW_ATTRIBUTE | KW_ASM);
2872         apply_modifiers(token->pos, &ctx);

2874         decl->ctype = ctx.ctype;
2875         decl->ctype.modifiers |= mod;
2876         decl->endpos = token->pos;

2878         /* Just a type declaration? */
2879         if (!ident) {
2880                 warning(token->pos, "missing identifier in declaration");
2881                 return expect(token, ';', "at the end of type declaration");
2882         }

2884         /* type define declaration? */
2885         is_typedef = ctx.storage_class == STypedef;

2887         /* Typedefs don't have meaningful storage */
2888         if (is_typedef)
2889                 decl->ctype.modifiers |= MOD_USERTYPE;

2891         bind_symbol(decl, ident, is_typedef ? NS_TYPEDEF: NS_SYMBOL);

2893         base_type = decl->ctype.base_type;

2895         if (is_typedef) {
2896                 if (base_type && !base_type->ident) {
2897                         switch (base_type->type) {
2898                         case SYM_STRUCT:
```

```
2899                        case SYM_UNION:
2900                        case SYM_ENUM:
2901                        case SYM_RESTRICT:
2902                                base_type->ident = ident;
2903                                break;
2904                        default:
2905                                break;
2906                        }
2907                }
2908        } else if (base_type && base_type->type == SYM_FN) {
2909                if (base_type->ctype.base_type == &incomplete_ctype) {
2910                        warning(decl->pos, "'%s()' has implicit return type",
2911                                show_ident(decl->ident));
2912                        base_type->ctype.base_type = &int_ctype;
2913                }
2914                /* K&R argument declaration? */
2915                if (lookup_type(token))
2916                        return parse_k_r_arguments(token, decl, list);
2917                if (match_op(token, '{'))
2918                        return parse_function_body(token, decl, list);

2920                if (!(decl->ctype.modifiers & MOD_STATIC))
2921                        decl->ctype.modifiers |= MOD_EXTERN;
2922        } else if (base_type == &void_ctype && !(decl->ctype.modifiers & MOD_EXT
2923                sparse_error(token->pos, "void declaration");
2924        }
2925        if (base_type == &incomplete_ctype) {
2926                warning(decl->pos, "'%s' has implicit type", show_ident(decl->id
2927                decl->ctype.base_type = &int_ctype;;
2928        }

2930        for (;;) {
2931                if (!is_typedef && match_op(token, '=')) {
2932                        token = initializer(&decl->initializer, token->next);
2933                }
2934                if (!is_typedef) {
2935                        if (validate_decl)
2936                                validate_decl(decl);

2938                        if (decl->initializer && decl->ctype.modifiers & MOD_EXT
2939                                warning(decl->pos, "symbol with external linkage
2940                                decl->ctype.modifiers &= ~MOD_EXTERN;
2941                        }

2943                        if (!(decl->ctype.modifiers & (MOD_EXTERN | MOD_INLINE))
2944                                add_symbol(list, decl);
2945                                fn_local_symbol(decl);
2946                        }
2947                }
2948                check_declaration(decl);
2949                if (decl->same_symbol) {
2950                        decl->definition = decl->same_symbol->definition;
2951                        decl->op = decl->same_symbol->op;
2952                }

2954                if (!match_op(token, ','))
2955                        break;

2957                token = token->next;
2958                ident = NULL;
2959                decl = alloc_symbol(token->pos, SYM_NODE);
2960                ctx.ctype = saved;
2961                token = handle_attributes(token, &ctx, KW_ATTRIBUTE);
2962                token = declarator(token, &ctx);
2963                token = handle_attributes(token, &ctx, KW_ATTRIBUTE | KW_ASM);
2964                apply_modifiers(token->pos, &ctx);
```



```
2965                decl->ctype = ctx.ctype;
2966                decl->ctype.modifiers |= mod;
2967                decl->endpos = token->pos;
2968                if (!ident) {
2969                        sparse_error(token->pos, "expected identifier name in ty
2970                        return token;
2971                }

2973                if (is_typedef)
2974                        decl->ctype.modifiers |= MOD_USERTYPE;

2976                bind_symbol(decl, ident, is_typedef ? NS_TYPEDEF: NS_SYMBOL);

2978                /* Function declarations are automatically extern unless specifi
2979                base_type = decl->ctype.base_type;
2980                if (!is_typedef && base_type && base_type->type == SYM_FN) {
2981                        if (!(decl->ctype.modifiers & MOD_STATIC))
2982                                decl->ctype.modifiers |= MOD_EXTERN;
2983                }
2984        }
2985        return expect(token, ';', "at end of declaration");
2986 }
```

```
**********************************************************
    1639 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/parse.dtd
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
<!ELEMENT parse (symbol+) >

<!ELEMENT symbol (symbol*) >

<!ATTLIST symbol type (uninitialized|preprocessor|basetype|node|pointer|function
                  id ID #REQUIRED
                  file CDATA #REQUIRED
                  start-line CDATA #REQUIRED
                  start-col CDATA #REQUIRED
                  end-line CDATA #IMPLIED
                  end-col CDATA #IMPLIED
                  end-file CDATA #IMPLIED

                  ident CDATA #IMPLIED
                  base-type IDREF #IMPLIED
                  base-type-builtin (char|signed char|unsigned char|short|signed

                  array-size CDATA #IMPLIED

                  bit-size CDATA #IMPLIED
                  alignment CDATA #IMPLIED
                  offset CDATA #IMPLIED
                  bit-offset CDATA #IMPLIED

                  auto (0|1) #IMPLIED
                  register (0|1) #IMPLIED
                  static (0|1) #IMPLIED
                  extern (0|1) #IMPLIED
                  const (0|1) #IMPLIED
                  volatile (0|1) #IMPLIED
                  signed (0|1) #IMPLIED
                  unsigned (0|1) #IMPLIED
                  char (0|1) #IMPLIED
                  short (0|1) #IMPLIED
                  long (0|1) #IMPLIED
                  long-long (0|1) #IMPLIED
                  typedef (0|1) #IMPLIED
                  inline (0|1) #IMPLIED
                  addressable (0|1) #IMPLIED
                  nocast (0|1) #IMPLIED
                  noderef (0|1) #IMPLIED
                  accessed (0|1) #IMPLIED
                  toplevel (0|1) #IMPLIED
                  label (0|1) #IMPLIED
                  assigned (0|1) #IMPLIED
                  type-type (0|1) #IMPLIED
                  safe (0|1) #IMPLIED
                  usertype (0|1) #IMPLIED
                  force (0|1) #IMPLIED
                  explicitly-signed (0|1) #IMPLIED
                  bitwise (0|1) #IMPLIED >
```

```
**********************************************************
    4632 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/parse.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef PARSE_H
#define PARSE_H
/*
 * Basic parsing data structures. Statements and symbols.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include "symbol.h"

enum statement_type {
        STMT_NONE,
        STMT_DECLARATION,
        STMT_EXPRESSION,
        STMT_COMPOUND,
        STMT_IF,
        STMT_RETURN,
        STMT_CASE,
        STMT_SWITCH,
        STMT_ITERATOR,
        STMT_LABEL,
        STMT_GOTO,
        STMT_ASM,
        STMT_CONTEXT,
        STMT_RANGE,
};

struct statement {
        enum statement_type type;
        struct position pos;
        struct statement *parent;
        union {
                struct /* declaration */ {
                        struct symbol_list *declaration;
                };
                struct /* context */ {
                        struct expression *expression;
                        struct expression *context;
                };
                struct /* return_statement */ {
                        struct expression *ret_value;
```



```
                        struct symbol *ret_target;
                };
                struct /* if_statement */ {
                        struct expression *if_conditional;
                        struct statement *if_true;
                        struct statement *if_false;
                };
                struct /* compound_struct */ {
                        struct statement_list *stmts;
                        struct symbol *ret;
                        struct symbol *inline_fn;
                        struct statement *args;
                };
                struct /* labeled_struct */ {
                        struct symbol *label_identifier;
                        struct statement *label_statement;
                };
                struct /* case_struct */ {
                        struct expression *case_expression;
                        struct expression *case_to;
                        struct statement *case_statement;
                        struct symbol *case_label;
                };
                struct /* switch_struct */ {
                        struct expression *switch_expression;
                        struct statement *switch_statement;
                        struct symbol *switch_break, *switch_case;
                };
                struct /* iterator_struct */ {
                        struct symbol *iterator_break;
                        struct symbol *iterator_continue;
                        struct symbol_list *iterator_syms;
                        struct statement  *iterator_pre_statement;
                        struct expression *iterator_pre_condition;

                        struct statement  *iterator_statement;

                        struct statement  *iterator_post_statement;
                        struct expression *iterator_post_condition;
                };
                struct /* goto_struct */ {
                        struct symbol *goto_label;

                        /* computed gotos have these: */
                        struct expression *goto_expression;
                        struct symbol_list *target_list;
                };
                struct /* asm */ {
                        struct expression *asm_string;
                        struct expression_list *asm_outputs;
                        struct expression_list *asm_inputs;
                        struct expression_list *asm_clobbers;
                        struct symbol_list *asm_labels;
                };
                struct /* range */ {
                        struct expression *range_expression;
                        struct expression *range_low;
                        struct expression *range_high;
                };
        };
};

extern struct symbol_list *function_computed_target_list;
extern struct statement_list *function_computed_goto_list;

extern struct token *parse_expression(struct token *, struct expression **);
```

```
127 extern struct symbol *label_symbol(struct token *token);

129 extern int show_statement(struct statement *);
130 extern void show_statement_list(struct statement_list *, const char *);
131 extern int show_expression(struct expression *);

133 typedef void (*validate_decl_t)(struct symbol *decl);
134 extern struct token *external_declaration(struct token *, struct symbol_list **,

136 extern struct symbol *ctype_integer(int size, int want_unsigned);

138 extern void copy_statement(struct statement *src, struct statement *dst);
139 extern int inline_function(struct expression *expr, struct symbol *sym);
140 extern void uninline(struct symbol *sym);
141 extern void init_parser(int);

143 static inline void stmt_set_parent_stmt(struct statement *stmt, struct statement
144 {
145         if (!stmt)
146                 return;
147         stmt->parent = parent;
148 }

150 static inline struct statement *stmt_get_parent_stmt(struct statement *stmt)
151 {
152         return stmt->parent;
153 }

155 #endif /* PARSE_H */
```

```
**********************************************************
   53472 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/pre-process.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Do C preprocessing, based on a token list gathered by
 * the tokenizer.
 *
 * This may not be the smartest preprocessor on the planet.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <limits.h>
#include <time.h>
#include <dirent.h>
#include <sys/stat.h>

#include "lib.h"
#include "allocate.h"
#include "parse.h"
#include "token.h"
#include "symbol.h"
#include "expression.h"
#include "scope.h"

static struct ident_list *macros;       // only needed for -dD
static int false_nesting = 0;
static int counter_macro = 0;           // __COUNTER__ expansion

#define INCLUDEPATHS 300
const char *includepath[INCLUDEPATHS+1] = {
        "",
        "/usr/include",
        "/usr/local/include",
        NULL
};
```



```
static const char **quote_includepath = includepath;
static const char **angle_includepath = includepath + 1;
static const char **isys_includepath   = includepath + 1;
static const char **sys_includepath   = includepath + 1;
static const char **dirafter_includepath = includepath + 3;

#define dirty_stream(stream)                            \
        do {                                            \
                if (!stream->dirty) {                   \
                        stream->dirty = 1;              \
                        if (!stream->ifndef)            \
                                stream->protect = NULL; \
                }                                       \
        } while(0)

#define end_group(stream)                                               \
        do {                                                            \
                if (stream->ifndef == stream->top_if) {                 \
                        stream->ifndef = NULL;                  \
                        if (!stream->dirty)                     \
                                stream->protect = NULL;         \
                        else if (stream->protect)               \
                                stream->dirty = 0;              \
                }                                                       \
        } while(0)

#define nesting_error(stream)           \
        do {                            \
                stream->dirty = 1;      \
                stream->ifndef = NULL;  \
                stream->protect = NULL; \
        } while(0)

static struct token *alloc_token(struct position *pos)
{
        struct token *token = __alloc_token(0);

        token->pos.stream = pos->stream;
        token->pos.line = pos->line;
        token->pos.pos = pos->pos;
        token->pos.whitespace = 1;
        return token;
}

/* Expand symbol 'sym' at '*list' */
static int expand(struct token **, struct symbol *);

static void replace_with_string(struct token *token, const char *str)
{
        int size = strlen(str) + 1;
        struct string *s = __alloc_string(size);

        s->length = size;
        memcpy(s->data, str, size);
        token_type(token) = TOKEN_STRING;
        token->string = s;
}

static void replace_with_integer(struct token *token, unsigned int val)
{
        char *buf = __alloc_bytes(11);
        sprintf(buf, "%u", val);
        token_type(token) = TOKEN_NUMBER;
        token->number = buf;
}
```

```
127 static struct symbol *lookup_macro(struct ident *ident)
128 {
129         struct symbol *sym = lookup_symbol(ident, NS_MACRO | NS_UNDEF);
130         if (sym && sym->namespace != NS_MACRO)
131                 sym = NULL;
132         return sym;
133 }

135 static int token_defined(struct token *token)
136 {
137         if (token_type(token) == TOKEN_IDENT) {
138                 struct symbol *sym = lookup_macro(token->ident);
139                 if (sym) {
140                         sym->used_in = file_scope;
141                         return 1;
142                 }
143                 return 0;
144         }

146         sparse_error(token->pos, "expected preprocessor identifier");
147         return 0;
148 }

150 static void replace_with_defined(struct token *token)
151 {
152         static const char *string[] = { "0", "1" };
153         int defined = token_defined(token);

155         token_type(token) = TOKEN_NUMBER;
156         token->number = string[defined];
157 }

159 static int expand_one_symbol(struct token **list)
160 {
161         struct token *token = *list;
162         struct symbol *sym;
163         static char buffer[12]; /* __DATE__: 3 + ' ' + 2 + ' ' + 4 + '\0' */
164         static time_t t = 0;

166         if (token->pos.noexpand)
167                 return 1;

169         sym = lookup_macro(token->ident);
170         if (sym) {
171                 store_macro_pos(token);
172                 sym->used_in = file_scope;
173                 return expand(list, sym);
174         }
175         if (token->ident == &__LINE___ident) {
176                 replace_with_integer(token, token->pos.line);
177         } else if (token->ident == &__FILE___ident) {
178                 replace_with_string(token, stream_name(token->pos.stream));
179         } else if (token->ident == &__DATE___ident) {
180                 if (!t)
181                         time(&t);
182                 strftime(buffer, 12, "%b %e %Y", localtime(&t));
183                 replace_with_string(token, buffer);
184         } else if (token->ident == &__TIME___ident) {
185                 if (!t)
186                         time(&t);
187                 strftime(buffer, 9, "%T", localtime(&t));
188                 replace_with_string(token, buffer);
189         } else if (token->ident == &__COUNTER___ident) {
190                 replace_with_integer(token, counter_macro++);
191         }
192         return 1;
```



```
193 }

195 static inline struct token *scan_next(struct token **where)
196 {
197         struct token *token = *where;
198         if (token_type(token) != TOKEN_UNTAINT)
199                 return token;
200         do {
201                 token->ident->tainted = 0;
202                 token = token->next;
203         } while (token_type(token) == TOKEN_UNTAINT);
204         *where = token;
205         return token;
206 }

208 static void expand_list(struct token **list)
209 {
210         struct token *next;
211         while (!eof_token(next = scan_next(list))) {
212                 if (token_type(next) != TOKEN_IDENT || expand_one_symbol(list))
213                         list = &next->next;
214         }
215 }

217 static void preprocessor_line(struct stream *stream, struct token **line);

219 static struct token *collect_arg(struct token *prev, int vararg, struct position
220 {
221         struct stream *stream = input_streams + prev->pos.stream;
222         struct token **p = &prev->next;
223         struct token *next;
224         int nesting = 0;

226         while (!eof_token(next = scan_next(p))) {
227                 if (next->pos.newline && match_op(next, '#')) {
228                         if (!next->pos.noexpand) {
229                                 sparse_error(next->pos,
230                                              "directive in argument list");
231                                 preprocessor_line(stream, p);
232                                 __free_token(next);     /* Free the '#' token */
233                                 continue;
234                         }
235                 }
236                 switch (token_type(next)) {
237                 case TOKEN_STREAMEND:
238                 case TOKEN_STREAMBEGIN:
239                         *p = &eof_token_entry;
240                         return next;
241                 case TOKEN_STRING:
242                 case TOKEN_WIDE_STRING:
243                         if (count > 1)
244                                 next->string->immutable = 1;
245                         break;
246                 }
247                 if (false_nesting) {
248                         *p = next->next;
249                         __free_token(next);
250                         continue;
251                 }
252                 if (match_op(next, '(')) {
253                         nesting++;
254                 } else if (match_op(next, ')')) {
255                         if (!nesting--)
256                                 break;
257                 } else if (match_op(next, ',') && !nesting && !vararg) {
258                         break;
```

```
259                 }
260                 next->pos.stream = pos->stream;
261                 next->pos.line = pos->line;
262                 next->pos.pos = pos->pos;
263                 p = &next->next;
264         }
265         *p = &eof_token_entry;
266         return next;
267 }

269 /*
270  * We store arglist as <counter> [arg1] <number of uses for arg1> ... eof
271  */

273 struct arg {
274         struct token *arg;
275         struct token *expanded;
276         struct token *str;
277         int n_normal;
278         int n_quoted;
279         int n_str;
280 };

282 static int collect_arguments(struct token *start, struct token *arglist, struct
283 {
284         int wanted = arglist->count.normal;
285         struct token *next = NULL;
286         int count = 0;

288         arglist = arglist->next;        /* skip counter */

290         if (!wanted) {
291                 next = collect_arg(start, 0, &what->pos, 0);
292                 if (eof_token(next))
293                         goto Eclosing;
294                 if (!eof_token(start->next) || !match_op(next, ')')) {
295                         count++;
296                         goto Emany;
297                 }
298         } else {
299                 for (count = 0; count < wanted; count++) {
300                         struct argcount *p = &arglist->next->count;
301                         next = collect_arg(start, p->vararg, &what->pos, p->norm
302                         if (eof_token(next))
303                                 goto Eclosing;
304                         if (p->vararg && wanted == 1 && eof_token(start->next))
305                                 break;
306                         arglist = arglist->next->next;
307                         args[count].arg = start->next;
308                         args[count].n_normal = p->normal;
309                         args[count].n_quoted = p->quoted;
310                         args[count].n_str = p->str;
311                         if (match_op(next, ')')) {
312                                 count++;
313                                 break;
314                         }
315                         start = next;
316                 }
317                 if (count == wanted && !match_op(next, ')'))
318                         goto Emany;
319                 if (count == wanted - 1) {
320                         struct argcount *p = &arglist->next->count;
321                         if (!p->vararg)
322                                 goto Efew;
323                         args[count].arg = NULL;
324                         args[count].n_normal = p->normal;
```



```
325                         args[count].n_quoted = p->quoted;
326                         args[count].n_str = p->str;
327                 }
328                 if (count < wanted - 1)
329                         goto Efew;
330         }
331         what->next = next->next;
332         return 1;

334 Efew:
335         sparse_error(what->pos, "macro \"%s\" requires %d arguments, but only %d
336                 show_token(what), wanted, count);
337         goto out;
338 Emany:
339         while (match_op(next, ',')) {
340                 next = collect_arg(next, 0, &what->pos, 0);
341                 count++;
342         }
343         if (eof_token(next))
344                 goto Eclosing;
345         sparse_error(what->pos, "macro \"%s\" passed %d arguments, but takes jus
346                 show_token(what), count, wanted);
347         goto out;
348 Eclosing:
349         sparse_error(what->pos, "unterminated argument list invoking macro \"%s\
350                 show_token(what));
351 out:
352         what->next = next->next;
353         return 0;
354 }

356 static struct token *dup_list(struct token *list)
357 {
358         struct token *res = NULL;
359         struct token **p = &res;

361         while (!eof_token(list)) {
362                 struct token *newtok = __alloc_token(0);
363                 *newtok = *list;
364                 *p = newtok;
365                 p = &newtok->next;
366                 list = list->next;
367         }
368         return res;
369 }

371 static const char *show_token_sequence(struct token *token, int quote)
372 {
373         static char buffer[MAX_STRING];
374         char *ptr = buffer;
375         int whitespace = 0;

377         if (!token && !quote)
378                 return "<none>";
379         while (!eof_token(token)) {
380                 const char *val = quote ? quote_token(token) : show_token(token)
381                 int len = strlen(val);

383                 if (ptr + whitespace + len >= buffer + sizeof(buffer)) {
384                         sparse_error(token->pos, "too long token expansion");
385                         break;
386                 }

388                 if (whitespace)
389                         *ptr++ = ' ';
390                 memcpy(ptr, val, len);
```

```
391                 ptr += len;
392                 token = token->next;
393                 whitespace = token->pos.whitespace;
394         }
395         *ptr = 0;
396         return buffer;
397 }

399 static struct token *stringify(struct token *arg)
400 {
401         const char *s = show_token_sequence(arg, 1);
402         int size = strlen(s)+1;
403         struct token *token = __alloc_token(0);
404         struct string *string = __alloc_string(size);

406         memcpy(string->data, s, size);
407         string->length = size;
408         token->pos = arg->pos;
409         token_type(token) = TOKEN_STRING;
410         token->string = string;
411         token->next = &eof_token_entry;
412         return token;
413 }

415 static void expand_arguments(int count, struct arg *args)
416 {
417         int i;
418         for (i = 0; i < count; i++) {
419                 struct token *arg = args[i].arg;
420                 if (!arg)
421                         arg = &eof_token_entry;
422                 if (args[i].n_str)
423                         args[i].str = stringify(arg);
424                 if (args[i].n_normal) {
425                         if (!args[i].n_quoted) {
426                                 args[i].expanded = arg;
427                                 args[i].arg = NULL;
428                         } else if (eof_token(arg)) {
429                                 args[i].expanded = arg;
430                         } else {
431                                 args[i].expanded = dup_list(arg);
432                         }
433                         expand_list(&args[i].expanded);
434                 }
435         }
436 }

438 /*
439  * Possibly valid combinations:
440  *  - ident + ident -> ident
441  *  - ident + number -> ident unless number contains '.', '+' or '-'.
442  *  - 'L' + char constant -> wide char constant
443  *  - 'L' + string literal -> wide string literal
444  *  - number + number -> number
445  *  - number + ident -> number
446  *  - number + '.' -> number
447  *  - number + '+' or '-' -> number, if number used to end on [eEpP].
448  *  - '.' + number -> number, if number used to start with a digit.
449  *  - special + special -> either special or an error.
450  */
451 static enum token_type combine(struct token *left, struct token *right, char *p)
452 {
453         int len;
454         enum token_type t1 = token_type(left), t2 = token_type(right);

456         if (t1 != TOKEN_IDENT && t1 != TOKEN_NUMBER && t1 != TOKEN_SPECIAL)
```



```
457                 return TOKEN_ERROR;

459         if (t1 == TOKEN_IDENT && left->ident == &L_ident) {
460                 if (t2 >= TOKEN_CHAR && t2 < TOKEN_WIDE_CHAR)
461                         return t2 + TOKEN_WIDE_CHAR - TOKEN_CHAR;
462                 if (t2 == TOKEN_STRING)
463                         return TOKEN_WIDE_STRING;
464         }

466         if (t2 != TOKEN_IDENT && t2 != TOKEN_NUMBER && t2 != TOKEN_SPECIAL)
467                 return TOKEN_ERROR;

469         strcpy(p, show_token(left));
470         strcat(p, show_token(right));
471         len = strlen(p);

473         if (len >= 256)
474                 return TOKEN_ERROR;

476         if (t1 == TOKEN_IDENT) {
477                 if (t2 == TOKEN_SPECIAL)
478                         return TOKEN_ERROR;
479                 if (t2 == TOKEN_NUMBER && strpbrk(p, "+-."))
480                         return TOKEN_ERROR;
481                 return TOKEN_IDENT;
482         }

484         if (t1 == TOKEN_NUMBER) {
485                 if (t2 == TOKEN_SPECIAL) {
486                         switch (right->special) {
487                         case '.':
488                                 break;
489                         case '+': case '-':
490                                 if (strchr("eEpP", p[len - 2]))
491                                         break;
492                         default:
493                                 return TOKEN_ERROR;
494                         }
495                 }
496                 return TOKEN_NUMBER;
497         }

499         if (p[0] == '.' && isdigit((unsigned char)p[1]))
500                 return TOKEN_NUMBER;

502         return TOKEN_SPECIAL;
503 }

505 static int merge(struct token *left, struct token *right)
506 {
507         static char buffer[512];
508         enum token_type res = combine(left, right, buffer);
509         int n;

511         switch (res) {
512         case TOKEN_IDENT:
513                 left->ident = built_in_ident(buffer);
514                 left->pos.noexpand = 0;
515                 return 1;

517         case TOKEN_NUMBER: {
518                 char *number = __alloc_bytes(strlen(buffer) + 1);
519                 memcpy(number, buffer, strlen(buffer) + 1);
520                 token_type(left) = TOKEN_NUMBER;        /* could be . + num */
521                 left->number = number;
522                 return 1;
```

```
523         }
525         case TOKEN_SPECIAL:
526                 if (buffer[2] && buffer[3])
527                         break;
528                 for (n = SPECIAL_BASE; n < SPECIAL_ARG_SEPARATOR; n++) {
529                         if (!memcmp(buffer, combinations[n-SPECIAL_BASE], 3)) {
530                                 left->special = n;
531                                 return 1;
532                         }
533                 }
534                 break;
536         case TOKEN_WIDE_CHAR:
537         case TOKEN_WIDE_STRING:
538                 token_type(left) = res;
539                 left->pos.noexpand = 0;
540                 left->string = right->string;
541                 return 1;
543         case TOKEN_WIDE_CHAR_EMBEDDED_0 ... TOKEN_WIDE_CHAR_EMBEDDED_3:
544                 token_type(left) = res;
545                 left->pos.noexpand = 0;
546                 memcpy(left->embedded, right->embedded, 4);
547                 return 1;
549         default:
550                 ;
551         }
552         sparse_error(left->pos, "'##' failed: concatenation is not a valid token
553         return 0;
554 }
556 static struct token *dup_token(struct token *token, struct position *streampos)
557 {
558         struct token *alloc = alloc_token(streampos);
559         token_type(alloc) = token_type(token);
560         alloc->pos.newline = token->pos.newline;
561         alloc->pos.whitespace = token->pos.whitespace;
562         alloc->number = token->number;
563         alloc->pos.noexpand = token->pos.noexpand;
564         return alloc;
565 }
567 static struct token **copy(struct token **where, struct token *list, int *count)
568 {
569         int need_copy = --*count;
570         while (!eof_token(list)) {
571                 struct token *token;
572                 if (need_copy)
573                         token = dup_token(list, &list->pos);
574                 else
575                         token = list;
576                 if (token_type(token) == TOKEN_IDENT && token->ident->tainted)
577                         token->pos.noexpand = 1;
578                 *where = token;
579                 where = &token->next;
580                 list = list->next;
581         }
582         *where = &eof_token_entry;
583         return where;
584 }
586 static int handle_kludge(struct token **p, struct arg *args)
587 {
588         struct token *t = (*p)->next->next;
```



```
589         while (1) {
590                 struct arg *v = &args[t->argnum];
591                 if (token_type(t->next) != TOKEN_CONCAT) {
592                         if (v->arg) {
593                                 /* ignore the first ## */
594                                 *p = (*p)->next;
595                                 return 0;
596                         }
597                         /* skip the entire thing */
598                         *p = t;
599                         return 1;
600                 }
601                 if (v->arg && !eof_token(v->arg))
602                         return 0; /* no magic */
603                 t = t->next->next;
604         }
605 }
607 static struct token **substitute(struct token **list, struct token *body, struct
608 {
609         struct position *base_pos = &(*list)->pos;
610         int *count;
611         enum {Normal, Placeholder, Concat} state = Normal;
613         for (; !eof_token(body); body = body->next) {
614                 struct token *added, *arg;
615                 struct token **tail;
616                 struct token *t;
618                 switch (token_type(body)) {
619                 case TOKEN_GNU_KLUDGE:
620                         /*
621                          * GNU kludge: if we had <comma>##<vararg>, behaviour
622                          * depends on whether we had enough arguments to have
623                          * a vararg.  If we did, ## is just ignored.  Otherwise
624                          * both , and ## are ignored.  Worse, there can be
625                          * an arbitrary number of ##<arg> in between; if all of
626                          * those are empty, we act as if they hadn't been there,
627                          * otherwise we act as if the kludge didn't exist.
628                          */
629                         t = body;
630                         if (handle_kludge(&body, args)) {
631                                 if (state == Concat)
632                                         state = Normal;
633                                 else
634                                         state = Placeholder;
635                                 continue;
636                         }
637                         added = dup_token(t, base_pos);
638                         token_type(added) = TOKEN_SPECIAL;
639                         tail = &added->next;
640                         break;
642                 case TOKEN_STR_ARGUMENT:
643                         arg = args[body->argnum].str;
644                         count = &args[body->argnum].n_str;
645                         goto copy_arg;
647                 case TOKEN_QUOTED_ARGUMENT:
648                         arg = args[body->argnum].arg;
649                         count = &args[body->argnum].n_quoted;
650                         if (!arg || eof_token(arg)) {
651                                 if (state == Concat)
652                                         state = Normal;
653                                 else
654                                         state = Placeholder;
```

```
655                             continue;
656                     }
657                     goto copy_arg;

659             case TOKEN_MACRO_ARGUMENT:
660                     arg = args[body->argnum].expanded;
661                     count = &args[body->argnum].n_normal;
662                     if (eof_token(arg)) {
663                             state = Normal;
664                             continue;
665                     }
666             copy_arg:
667                     tail = copy(&added, arg, count);
668                     added->pos.newline = body->pos.newline;
669                     added->pos.whitespace = body->pos.whitespace;
670                     break;

672             case TOKEN_CONCAT:
673                     if (state == Placeholder)
674                             state = Normal;
675                     else
676                             state = Concat;
677                     continue;

679             case TOKEN_IDENT:
680                     added = dup_token(body, base_pos);
681                     if (added->ident->tainted)
682                             added->pos.noexpand = 1;
683                     tail = &added->next;
684                     break;

686             default:
687                     added = dup_token(body, base_pos);
688                     tail = &added->next;
689                     break;
690             }

692             /*
693              * if we got to doing real concatenation, we already have
694              * added something into the list, so containing_token() is OK.
695              */
696             if (state == Concat && merge(containing_token(list), added)) {
697                     *list = added->next;
698                     if (tail != &added->next)
699                             list = tail;
700             } else {
701                     *list = added;
702                     list = tail;
703             }
704             state = Normal;
705     }
706     *list = &eof_token_entry;
707     return list;
708 }

710 static int expand(struct token **list, struct symbol *sym)
711 {
712     struct token *last;
713     struct token *token = *list;
714     struct ident *expanding = token->ident;
715     struct token **tail;
716     int nargs = sym->arglist ? sym->arglist->count.normal : 0;
717     struct arg args[nargs];

719     if (expanding->tainted) {
720             token->pos.noexpand = 1;
```



```
721             return 1;
722     }

724     if (sym->arglist) {
725             if (!match_op(scan_next(&token->next), '('))
726                     return 1;
727             if (!collect_arguments(token->next, sym->arglist, args, token))
728                     return 1;
729             expand_arguments(nargs, args);
730     }

732     expanding->tainted = 1;

734     last = token->next;
735     tail = substitute(list, sym->expansion, args);
736     /*
737      * Note that it won't be eof - at least TOKEN_UNTAINT will be there.
738      * We still can lose the newline flag if the sucker expands to nothing,
739      * but the price of dealing with that is probably too high (we'd need
740      * to collect the flags during scan_next())
741      */
742     (*list)->pos.newline = token->pos.newline;
743     (*list)->pos.whitespace = token->pos.whitespace;
744     *tail = last;

746     return 0;
747 }

749 static const char *token_name_sequence(struct token *token, int endop, struct to
750 {
751     static char buffer[256];
752     char *ptr = buffer;

754     while (!eof_token(token) && !match_op(token, endop)) {
755             int len;
756             const char *val = token->string->data;
757             if (token_type(token) != TOKEN_STRING)
758                     val = show_token(token);
759             len = strlen(val);
760             memcpy(ptr, val, len);
761             ptr += len;
762             token = token->next;
763     }
764     *ptr = 0;
765     if (endop && !match_op(token, endop))
766             sparse_error(start->pos, "expected '>' at end of filename");
767     return buffer;
768 }

770 static int already_tokenized(const char *path)
771 {
772     int stream, next;

774     for (stream = *hash_stream(path); stream >= 0 ; stream = next) {
775             struct stream *s = input_streams + stream;

777             next = s->next_stream;
778             if (s->once) {
779                     if (strcmp(path, s->name))
780                             continue;
781                     return 1;
782             }
783             if (s->constant != CONSTANT_FILE_YES)
784                     continue;
785             if (strcmp(path, s->name))
786                     continue;
```

```
787                 if (s->protect && !lookup_macro(s->protect))
788                         continue;
789                 return 1;
790         }
791         return 0;
792 }

794 /* Handle include of header files.
795  * The relevant options are made compatible with gcc. The only options that
796  * are not supported is -withprefix and friends.
797  *
798  * Three set of include paths are known:
799  * quote_includepath:   Path to search when using #include "file.h"
800  * angle_includepath:   Paths to search when using #include <file.h>
801  * isys_includepath:    Paths specified with -isystem, come before the
802  *                      built-in system include paths. Gcc would suppress
803  *                      warnings from system headers. Here we separate
804  *                      them from the angle_ ones to keep search ordering.
805  *
806  * sys_includepath:     Built-in include paths.
807  * dirafter_includepath Paths added with -dirafter.
808  *
809  * The above is implemented as one array with pointers
810  *                          +--------------+
811  * quote_includepath --->   |              |
812  *                          +--------------+
813  *                          |              |
814  *                          +--------------+
815  * angle_includepath --->   |              |
816  *                          +--------------+
817  * isys_includepath  --->   |              |
818  *                          +--------------+
819  * sys_includepath   --->   |              |
820  *                          +--------------+
821  * dirafter_includepath ->  |              |
822  *                          +--------------+
823  *
824  * -I dir insert dir just before isys_includepath and move the rest
825  * -I- makes all dirs specified with -I before to quote dirs only and
826  *   angle_includepath is set equal to isys_includepath.
827  * -nostdinc removes all sys dirs by storing NULL in entry pointed
828  *   to by * sys_includepath. Note that this will reset all dirs built-in
829  *   and added before -nostdinc by -isystem and -idirafter.
830  * -isystem dir adds dir where isys_includepath points adding this dir as
831  *   first systemdir
832  * -idirafter dir adds dir to the end of the list
833  */

835 static void set_stream_include_path(struct stream *stream)
836 {
837         const char *path = stream->path;
838         if (!path) {
839                 const char *p = strrchr(stream->name, '/');
840                 path = "";
841                 if (p) {
842                         int len = p - stream->name + 1;
843                         char *m = malloc(len+1);
844                         /* This includes the final "/" */
845                         memcpy(m, stream->name, len);
846                         m[len] = 0;
847                         path = m;
848                 }
849                 stream->path = path;
850         }
851         includepath[0] = path;
852 }
```



```
854 static int try_include(const char *path, const char *filename, int flen, struct
855 {
856         int fd;
857         int plen = strlen(path);
858         static char fullname[PATH_MAX];

860         memcpy(fullname, path, plen);
861         if (plen && path[plen-1] != '/') {
862                 fullname[plen] = '/';
863                 plen++;
864         }
865         memcpy(fullname+plen, filename, flen);
866         if (already_tokenized(fullname))
867                 return 1;
868         fd = open(fullname, O_RDONLY);
869         if (fd >= 0) {
870                 char * streamname = __alloc_bytes(plen + flen);
871                 memcpy(streamname, fullname, plen + flen);
872                 *where = tokenize(streamname, fd, *where, next_path);
873                 close(fd);
874                 return 1;
875         }
876         return 0;
877 }

879 static int do_include_path(const char **pptr, struct token **list, struct token
880 {
881         const char *path;

883         while ((path = *pptr++) != NULL) {
884                 if (!try_include(path, filename, flen, list, pptr))
885                         continue;
886                 return 1;
887         }
888         return 0;
889 }

891 static int free_preprocessor_line(struct token *token)
892 {
893         while (token_type(token) != TOKEN_EOF) {
894                 struct token *free = token;
895                 token = token->next;
896                 __free_token(free);
897         };
898         return 1;
899 }

901 const char *find_include(const char *skip, const char *look_for)
902 {
903         DIR *dp;
904         struct dirent *entry;
905         struct stat statbuf;
906         const char *ret;
907         char cwd[PATH_MAX];
908         static char buf[PATH_MAX + 1];

910         dp = opendir(".");
911         if (!dp)
912                 return NULL;

914         if (!getcwd(cwd, sizeof(cwd)))
915                 return NULL;

917         while ((entry = readdir(dp))) {
918                 lstat(entry->d_name, &statbuf);
```

```
920                 if (strcmp(entry->d_name, look_for) == 0) {
921                         snprintf(buf, sizeof(buf), "%s/%s", cwd, entry->d_name);
922                         return buf;
923                 }
924
925                 if (S_ISDIR(statbuf.st_mode)) {
926                         /* Found a directory, but ignore . and .. */
927                         if (strcmp(".", entry->d_name) == 0 ||
928                             strcmp("..", entry->d_name) == 0 ||
929                             strcmp(skip, entry->d_name) == 0)
930                                 continue;
931
932                         chdir(entry->d_name);
933                         ret = find_include("", look_for);
934                         chdir("..");
935                         if (ret)
936                                 return ret;
937                 }
938         }
939         closedir(dp);
940
941         return NULL;
942 }
943
944 const char *search_dir(const char *stop, const char *look_for)
945 {
946         char cwd[PATH_MAX];
947         int len;
948         const char *ret;
949         int cnt = 0;
950
951         if (!getcwd(cwd, sizeof(cwd)))
952                 return NULL;
953
954         len = strlen(cwd);
955         while (len >= 0) {
956                 ret = find_include(cnt++ ? cwd + len + 1 : "", look_for);
957                 if (ret)
958                         return ret;
959
960                 if (strcmp(cwd, stop) == 0 ||
961                     strcmp(cwd, "/usr/include") == 0 ||
962                     strcmp(cwd, "/usr/local/include") == 0 ||
963                     strlen(cwd) <= 10 ||  /* heck...  don't search /usr/lib/ */
964                     strcmp(cwd, "/") == 0)
965                         return NULL;
966
967                 while (--len >= 0) {
968                         if (cwd[len] == '/') {
969                                 cwd[len] = '\0';
970                                 break;
971                         }
972                 }
973
974                 chdir("..");
975         }
976         return NULL;
977 }
978
979 static void use_best_guess_header_file(struct token *token, const char *filename
980 {
981         char cwd[PATH_MAX];
982         char dir_part[PATH_MAX];
983         const char *file_part;
984         const char *include_name;
```



```
985         int len;
986
987         if (!filename || filename[0] == '\0')
988                 return;
989
990         file_part = filename;
991         while ((filename = strchr(filename, '/'))) {
992                 ++filename;
993                 if (filename[0])
994                         file_part = filename;
995         }
996
997         snprintf(dir_part, sizeof(dir_part), "%s", stream_name(token->pos.stream
998         len = strlen(dir_part);
999         while (--len >= 0) {
1000                if (dir_part[len] == '/') {
1001                        dir_part[len] = '\0';
1002                        break;
1003                }
1004        }
1005        if (len < 0)
1006                sprintf(dir_part, ".");
1007
1008        if (!getcwd(cwd, sizeof(cwd)))
1009                return;
1010
1011        chdir(dir_part);
1012        include_name = search_dir(cwd, file_part);
1013        chdir(cwd);
1014        if (!include_name)
1015                return;
1016        sparse_error(token->pos, "using '%s'", include_name);
1017
1018        try_include("", include_name, strlen(include_name), list, includepath);
1019 }
1020
1021 static int handle_include_path(struct stream *stream, struct token **list, struc
1022 {
1023         const char *filename;
1024         struct token *next;
1025         const char **path;
1026         int expect;
1027         int flen;
1028
1029         next = token->next;
1030         expect = '>';
1031         if (!match_op(next, '<')) {
1032                 expand_list(&token->next);
1033                 expect = 0;
1034                 next = token;
1035                 if (match_op(token->next, '<')) {
1036                         next = token->next;
1037                         expect = '>';
1038                 }
1039         }
1040
1041         token = next->next;
1042         filename = token_name_sequence(token, expect, token);
1043         flen = strlen(filename) + 1;
1044
1045         /* Absolute path? */
1046         if (filename[0] == '/') {
1047                 if (try_include("", filename, flen, list, includepath))
1048                         return 0;
1049                 goto out;
1050         }
```

```
1052         switch (how) {
1053         case 1:
1054                 path = stream->next_path;
1055                 break;
1056         case 2:
1057                 includepath[0] = "";
1058                 path = includepath;
1059                 break;
1060         default:
1061                 /* Dir of input file is first dir to search for quoted includes
1062                 set_stream_include_path(stream);
1063                 path = expect ? angle_includepath : quote_includepath;
1064                 break;
1065         }
1066         /* Check the standard include paths.. */
1067         if (do_include_path(path, list, token, filename, flen))
1068                 return 0;
1069 out:
1070         sparse_error(token->pos, "unable to open '%s'", filename);
1071         use_best_guess_header_file(token, filename, list);
1072         return 0;
1073 }

1075 static int handle_include(struct stream *stream, struct token **list, struct tok
1076 {
1077         return handle_include_path(stream, list, token, 0);
1078 }

1080 static int handle_include_next(struct stream *stream, struct token **list, struc
1081 {
1082         return handle_include_path(stream, list, token, 1);
1083 }

1085 static int handle_argv_include(struct stream *stream, struct token **list, struc
1086 {
1087         return handle_include_path(stream, list, token, 2);
1088 }

1090 static int token_different(struct token *t1, struct token *t2)
1091 {
1092         int different;

1094         if (token_type(t1) != token_type(t2))
1095                 return 1;

1097         switch (token_type(t1)) {
1098         case TOKEN_IDENT:
1099                 different = t1->ident != t2->ident;
1100                 break;
1101         case TOKEN_ARG_COUNT:
1102         case TOKEN_UNTAINT:
1103         case TOKEN_CONCAT:
1104         case TOKEN_GNU_KLUDGE:
1105                 different = 0;
1106                 break;
1107         case TOKEN_NUMBER:
1108                 different = strcmp(t1->number, t2->number);
1109                 break;
1110         case TOKEN_SPECIAL:
1111                 different = t1->special != t2->special;
1112                 break;
1113         case TOKEN_MACRO_ARGUMENT:
1114         case TOKEN_QUOTED_ARGUMENT:
1115         case TOKEN_STR_ARGUMENT:
1116                 different = t1->argnum != t2->argnum;
```



```
1117                 break;
1118         case TOKEN_CHAR_EMBEDDED_0 ... TOKEN_CHAR_EMBEDDED_3:
1119         case TOKEN_WIDE_CHAR_EMBEDDED_0 ... TOKEN_WIDE_CHAR_EMBEDDED_3:
1120                 different = memcmp(t1->embedded, t2->embedded, 4);
1121                 break;
1122         case TOKEN_CHAR:
1123         case TOKEN_WIDE_CHAR:
1124         case TOKEN_STRING:
1125         case TOKEN_WIDE_STRING: {
1126                 struct string *s1, *s2;

1128                 s1 = t1->string;
1129                 s2 = t2->string;
1130                 different = 1;
1131                 if (s1->length != s2->length)
1132                         break;
1133                 different = memcmp(s1->data, s2->data, s1->length);
1134                 break;
1135         }
1136         default:
1137                 different = 1;
1138                 break;
1139         }
1140         return different;
1141 }

1143 static int token_list_different(struct token *list1, struct token *list2)
1144 {
1145         for (;;) {
1146                 if (list1 == list2)
1147                         return 0;
1148                 if (!list1 || !list2)
1149                         return 1;
1150                 if (token_different(list1, list2))
1151                         return 1;
1152                 list1 = list1->next;
1153                 list2 = list2->next;
1154         }
1155 }

1157 static inline void set_arg_count(struct token *token)
1158 {
1159         token_type(token) = TOKEN_ARG_COUNT;
1160         token->count.normal = token->count.quoted =
1161         token->count.str = token->count.vararg = 0;
1162 }

1164 static struct token *parse_arguments(struct token *list)
1165 {
1166         struct token *arg = list->next, *next = list;
1167         struct argcount *count = &list->count;

1169         set_arg_count(list);

1171         if (match_op(arg, ')')) {
1172                 next = arg->next;
1173                 list->next = &eof_token_entry;
1174                 return next;
1175         }

1177         while (token_type(arg) == TOKEN_IDENT) {
1178                 if (arg->ident == &__VA_ARGS___ident)
1179                         goto Eva_args;
1180                 if (!++count->normal)
1181                         goto Eargs;
1182                 next = arg->next;
```

```
1184                 if (match_op(next, ','')) {
1185                         set_arg_count(next);
1186                         arg = next->next;
1187                         continue;
1188                 }

1190                 if (match_op(next, ')')) {
1191                         set_arg_count(next);
1192                         next = next->next;
1193                         arg->next->next = &eof_token_entry;
1194                         return next;
1195                 }

1197                 /* normal cases are finished here */

1199                 if (match_op(next, SPECIAL_ELLIPSIS)) {
1200                         if (match_op(next->next, ')')) {
1201                                 set_arg_count(next);
1202                                 next->count.vararg = 1;
1203                                 next = next->next;
1204                                 arg->next->next = &eof_token_entry;
1205                                 return next->next;
1206                         }

1208                         arg = next;
1209                         goto Enotclosed;
1210                 }

1212                 if (eof_token(next)) {
1213                         goto Enotclosed;
1214                 } else {
1215                         arg = next;
1216                         goto Ebadstuff;
1217                 }
1218         }

1220         if (match_op(arg, SPECIAL_ELLIPSIS)) {
1221                 next = arg->next;
1222                 token_type(arg) = TOKEN_IDENT;
1223                 arg->ident = &__VA_ARGS__ident;
1224                 if (!match_op(next, ')'))
1225                         goto Enotclosed;
1226                 if (!++count->normal)
1227                         goto Eargs;
1228                 set_arg_count(next);
1229                 next->count.vararg = 1;
1230                 next = next->next;
1231                 arg->next->next = &eof_token_entry;
1232                 return next;
1233         }

1235         if (eof_token(arg)) {
1236                 arg = next;
1237                 goto Enotclosed;
1238         }
1239         if (match_op(arg, ','))
1240                 goto Emissing;
1241         else
1242                 goto Ebadstuff;


1245 Emissing:
1246         sparse_error(arg->pos, "parameter name missing");
1247         return NULL;
1248 Ebadstuff:
```



```
1249         sparse_error(arg->pos, "\"%s\" may not appear in macro parameter list",
1250                 show_token(arg));
1251         return NULL;
1252 Enotclosed:
1253         sparse_error(arg->pos, "missing ')' in macro parameter list");
1254         return NULL;
1255 Eva_args:
1256         sparse_error(arg->pos, "__VA_ARGS__ can only appear in the expansion of
1257         return NULL;
1258 Eargs:
1259         sparse_error(arg->pos, "too many arguments in macro definition");
1260         return NULL;
1261 }

1263 static int try_arg(struct token *token, enum token_type type, struct token *argl
1264 {
1265         struct ident *ident = token->ident;
1266         int nr;

1268         if (!arglist || token_type(token) != TOKEN_IDENT)
1269                 return 0;

1271         arglist = arglist->next;

1273         for (nr = 0; !eof_token(arglist); nr++, arglist = arglist->next->next) {
1274                 if (arglist->ident == ident) {
1275                         struct argcount *count = &arglist->next->count;
1276                         int n;

1278                         token->argnum = nr;
1279                         token_type(token) = type;
1280                         switch (type) {
1281                         case TOKEN_MACRO_ARGUMENT:
1282                                 n = ++count->normal;
1283                                 break;
1284                         case TOKEN_QUOTED_ARGUMENT:
1285                                 n = ++count->quoted;
1286                                 break;
1287                         default:
1288                                 n = ++count->str;
1289                         }
1290                         if (n)
1291                                 return count->vararg ? 2 : 1;
1292                         /*
1293                          * XXX - need saner handling of that
1294                          * (>= 1024 instances of argument)
1295                          */
1296                         token_type(token) = TOKEN_ERROR;
1297                         return -1;
1298                 }
1299         }
1300         return 0;
1301 }

1303 static struct token *handle_hash(struct token **p, struct token *arglist)
1304 {
1305         struct token *token = *p;
1306         if (arglist) {
1307                 struct token *next = token->next;
1308                 if (!try_arg(next, TOKEN_STR_ARGUMENT, arglist))
1309                         goto Equote;
1310                 next->pos.whitespace = token->pos.whitespace;
1311                 __free_token(token);
1312                 token = *p = next;
1313         } else {
1314                 token->pos.noexpand = 1;
```

```
1315         }
1316         return token;

1318 Equote:
1319         sparse_error(token->pos, "'#' is not followed by a macro parameter");
1320         return NULL;
1321 }

1323 /* token->next is ## */
1324 static struct token *handle_hashhash(struct token *token, struct token *arglist)
1325 {
1326         struct token *last = token;
1327         struct token *concat;
1328         int state = match_op(token, ',');
1329
1330         try_arg(token, TOKEN_QUOTED_ARGUMENT, arglist);

1332         while (1) {
1333                 struct token *t;
1334                 int is_arg;

1336                 /* eat duplicate ## */
1337                 concat = token->next;
1338                 while (match_op(t = concat->next, SPECIAL_HASHHASH)) {
1339                         token->next = t;
1340                         __free_token(concat);
1341                         concat = t;
1342                 }
1343                 token_type(concat) = TOKEN_CONCAT;

1345                 if (eof_token(t))
1346                         goto Econcat;

1348                 if (match_op(t, '#')) {
1349                         t = handle_hash(&concat->next, arglist);
1350                         if (!t)
1351                                 return NULL;
1352                 }

1354                 is_arg = try_arg(t, TOKEN_QUOTED_ARGUMENT, arglist);

1356                 if (state == 1 && is_arg) {
1357                         state = is_arg;
1358                 } else {
1359                         last = t;
1360                         state = match_op(t, ',');
1361                 }

1363                 token = t;
1364                 if (!match_op(token->next, SPECIAL_HASHHASH))
1365                         break;
1366         }
1367         /* handle GNU ,##__VA_ARGS__ kludge, in all its weirdness */
1368         if (state == 2)
1369                 token_type(last) = TOKEN_GNU_KLUDGE;
1370         return token;

1372 Econcat:
1373         sparse_error(concat->pos, "'##' cannot appear at the ends of macro expan
1374         return NULL;
1375 }

1377 static struct token *parse_expansion(struct token *expansion, struct token *argl
1378 {
1379         struct token *token = expansion;
1380         struct token **p;
```



```
1382         if (match_op(token, SPECIAL_HASHHASH))
1383                 goto Econcat;

1385         for (p = &expansion; !eof_token(token); p = &token->next, token = *p) {
1386                 if (match_op(token, '#')) {
1387                         token = handle_hash(p, arglist);
1388                         if (!token)
1389                                 return NULL;
1390                 }
1391                 if (match_op(token->next, SPECIAL_HASHHASH)) {
1392                         token = handle_hashhash(token, arglist);
1393                         if (!token)
1394                                 return NULL;
1395                 } else {
1396                         try_arg(token, TOKEN_MACRO_ARGUMENT, arglist);
1397                 }
1398                 switch (token_type(token)) {
1399                 case TOKEN_ERROR:
1400                         goto Earg;

1402                 case TOKEN_STRING:
1403                 case TOKEN_WIDE_STRING:
1404                         token->string->immutable = 1;
1405                         break;
1406                 }
1407         }
1408         token = alloc_token(&expansion->pos);
1409         token_type(token) = TOKEN_UNTAINT;
1410         token->ident = name;
1411         token->next = *p;
1412         *p = token;
1413         return expansion;

1415 Econcat:
1416         sparse_error(token->pos, "'##' cannot appear at the ends of macro expans
1417         return NULL;
1418 Earg:
1419         sparse_error(token->pos, "too many instances of argument in body");
1420         return NULL;
1421 }

1423 static int do_handle_define(struct stream *stream, struct token **line, struct t
1424 {
1425         struct token *arglist, *expansion;
1426         struct token *left = token->next;
1427         struct symbol *sym;
1428         struct ident *name;
1429         int ret;

1431         if (token_type(left) != TOKEN_IDENT) {
1432                 sparse_error(token->pos, "expected identifier to 'define'");
1433                 return 1;
1434         }

1436         name = left->ident;

1438         arglist = NULL;
1439         expansion = left->next;
1440         if (!expansion->pos.whitespace) {
1441                 if (match_op(expansion, '(')) {
1442                         arglist = expansion;
1443                         expansion = parse_arguments(expansion);
1444                         if (!expansion)
1445                                 return 1;
1446                 } else if (!eof_token(expansion)) {
```

```
1447                         warning(expansion->pos,
1448                                 "no whitespace before object-like macro body");
1449                 }
1450         }

1452         expansion = parse_expansion(expansion, arglist, name);
1453         if (!expansion)
1454                 return 1;

1456         ret = 1;
1457         sym = lookup_symbol(name, NS_MACRO | NS_UNDEF);
1458         if (sym) {
1459                 int clean;

1461                 if (attr < sym->attr)
1462                         goto out;

1464                 clean = (attr == sym->attr && sym->namespace == NS_MACRO);

1466                 if (token_list_different(sym->expansion, expansion) ||
1467                     token_list_different(sym->arglist, arglist)) {
1468                         ret = 0;
1469                         if ((clean && attr == SYM_ATTR_NORMAL)
1470                                         || sym->used_in == file_scope) {
1471                                 warning(left->pos, "preprocessor token %.*s rede
1472                                                 name->len, name->name);
1473                                 info(sym->pos, "this was the original definition
1474                         }
1475                 } else if (clean)
1476                         goto out;
1477         }

1479         if (!sym || sym->scope != file_scope) {
1480                 sym = alloc_symbol(left->pos, SYM_NODE);
1481                 bind_symbol(sym, name, NS_MACRO);
1482                 add_ident(&macros, name);
1483                 ret = 0;
1484         }

1486         if (!ret) {
1487                 sym->expansion = expansion;
1488                 sym->arglist = arglist;
1489                 __free_token(token);    /* Free the "define" token, but not the
1490         }

1492         sym->namespace = NS_MACRO;
1493         sym->used_in = NULL;
1494         sym->attr = attr;
1495 out:
1496         return ret;
1497 }

1499 static int handle_define(struct stream *stream, struct token **line, struct toke
1500 {
1501         return do_handle_define(stream, line, token, SYM_ATTR_NORMAL);
1502 }

1504 static int handle_weak_define(struct stream *stream, struct token **line, struct
1505 {
1506         return do_handle_define(stream, line, token, SYM_ATTR_WEAK);
1507 }

1509 static int handle_strong_define(struct stream *stream, struct token **line, stru
1510 {
1511         return do_handle_define(stream, line, token, SYM_ATTR_STRONG);
1512 }
```



```
1514 static int do_handle_undef(struct stream *stream, struct token **line, struct to
1515 {
1516         struct token *left = token->next;
1517         struct symbol *sym;

1519         if (token_type(left) != TOKEN_IDENT) {
1520                 sparse_error(token->pos, "expected identifier to 'undef'");
1521                 return 1;
1522         }

1524         sym = lookup_symbol(left->ident, NS_MACRO | NS_UNDEF);
1525         if (sym) {
1526                 if (attr < sym->attr)
1527                         return 1;
1528                 if (attr == sym->attr && sym->namespace == NS_UNDEF)
1529                         return 1;
1530         } else if (attr <= SYM_ATTR_NORMAL)
1531                 return 1;

1533         if (!sym || sym->scope != file_scope) {
1534                 sym = alloc_symbol(left->pos, SYM_NODE);
1535                 bind_symbol(sym, left->ident, NS_MACRO);
1536         }

1538         sym->namespace = NS_UNDEF;
1539         sym->used_in = NULL;
1540         sym->attr = attr;

1542         return 1;
1543 }

1545 static int handle_undef(struct stream *stream, struct token **line, struct token
1546 {
1547         return do_handle_undef(stream, line, token, SYM_ATTR_NORMAL);
1548 }

1550 static int handle_strong_undef(struct stream *stream, struct token **line, struc
1551 {
1552         return do_handle_undef(stream, line, token, SYM_ATTR_STRONG);
1553 }

1555 static int preprocessor_if(struct stream *stream, struct token *token, int true)
1556 {
1557         token_type(token) = false_nesting ? TOKEN_SKIP_GROUPS : TOKEN_IF;
1558         free_preprocessor_line(token->next);
1559         token->next = stream->top_if;
1560         stream->top_if = token;
1561         if (false_nesting || true != 1)
1562                 false_nesting++;
1563         return 0;
1564 }

1566 static int handle_ifdef(struct stream *stream, struct token **line, struct token
1567 {
1568         struct token *next = token->next;
1569         int arg;
1570         if (token_type(next) == TOKEN_IDENT) {
1571                 arg = token_defined(next);
1572         } else {
1573                 dirty_stream(stream);
1574                 if (!false_nesting)
1575                         sparse_error(token->pos, "expected preprocessor identifi
1576                 arg = -1;
1577         }
1578         return preprocessor_if(stream, token, arg);
```

```
}

static int handle_ifndef(struct stream *stream, struct token **line, struct toke
{
        struct token *next = token->next;
        int arg;
        if (token_type(next) == TOKEN_IDENT) {
                if (!stream->dirty && !stream->ifndef) {
                        if (!stream->protect) {
                                stream->ifndef = token;
                                stream->protect = next->ident;
                        } else if (stream->protect == next->ident) {
                                stream->ifndef = token;
                                stream->dirty = 1;
                        }
                }
                arg = !token_defined(next);
        } else {
                dirty_stream(stream);
                if (!false_nesting)
                        sparse_error(token->pos, "expected preprocessor identifi
                arg = -1;
        }

        return preprocessor_if(stream, token, arg);
}

static const char *show_token_sequence(struct token *token, int quote);

/*
 * Expression handling for #if and #elif; it differs from normal expansion
 * due to special treatment of "defined".
 */
static int expression_value(struct token **where)
{
        struct expression *expr;
        struct token *p;
        struct token **list = where, **beginning = NULL;
        long long value;
        int state = 0;

        while (!eof_token(p = scan_next(list))) {
                switch (state) {
                case 0:
                        if (token_type(p) != TOKEN_IDENT)
                                break;
                        if (p->ident == &defined_ident) {
                                state = 1;
                                beginning = list;
                                break;
                        }
                        if (!expand_one_symbol(list))
                                continue;
                        if (token_type(p) != TOKEN_IDENT)
                                break;
                        token_type(p) = TOKEN_ZERO_IDENT;
                        break;
                case 1:
                        if (match_op(p, '(')) {
                                state = 2;
                        } else {
                                state = 0;
                                replace_with_defined(p);
                                *beginning = p;
                        }
                        break;
```



```
                case 2:
                        if (token_type(p) == TOKEN_IDENT)
                                state = 3;
                        else
                                state = 0;
                        replace_with_defined(p);
                        *beginning = p;
                        break;
                case 3:
                        state = 0;
                        if (!match_op(p, ')'))
                                sparse_error(p->pos, "missing ')' after \"define
                        *list = p->next;
                        continue;
                }
                list = &p->next;
        }

        p = constant_expression(*where, &expr);
        if (!eof_token(p))
                sparse_error(p->pos, "garbage at end: %s", show_token_sequence(p
        value = get_expression_value(expr);
        return value != 0;
}

static int handle_if(struct stream *stream, struct token **line, struct token *t
{
        int value = 0;
        if (!false_nesting)
                value = expression_value(&token->next);

        dirty_stream(stream);
        return preprocessor_if(stream, token, value);
}

static int handle_elif(struct stream * stream, struct token **line, struct token
{
        struct token *top_if = stream->top_if;
        end_group(stream);

        if (!top_if) {
                nesting_error(stream);
                sparse_error(token->pos, "unmatched #elif within stream");
                return 1;
        }

        if (token_type(top_if) == TOKEN_ELSE) {
                nesting_error(stream);
                sparse_error(token->pos, "#elif after #else");
                if (!false_nesting)
                        false_nesting = 1;
                return 1;
        }

        dirty_stream(stream);
        if (token_type(top_if) != TOKEN_IF)
                return 1;
        if (false_nesting) {
                false_nesting = 0;
                if (!expression_value(&token->next))
                        false_nesting = 1;
        } else {
                false_nesting = 1;
                token_type(top_if) = TOKEN_SKIP_GROUPS;
        }
        return 1;
```

```
}

static int handle_else(struct stream *stream, struct token **line, struct token
{
        struct token *top_if = stream->top_if;
        end_group(stream);

        if (!top_if) {
                nesting_error(stream);
                sparse_error(token->pos, "unmatched #else within stream");
                return 1;
        }

        if (token_type(top_if) == TOKEN_ELSE) {
                nesting_error(stream);
                sparse_error(token->pos, "#else after #else");
        }
        if (false_nesting) {
                if (token_type(top_if) == TOKEN_IF)
                        false_nesting = 0;
        } else {
                false_nesting = 1;
        }
        token_type(top_if) = TOKEN_ELSE;
        return 1;
}

static int handle_endif(struct stream *stream, struct token **line, struct token
{
        struct token *top_if = stream->top_if;
        end_group(stream);
        if (!top_if) {
                nesting_error(stream);
                sparse_error(token->pos, "unmatched #endif in stream");
                return 1;
        }
        if (false_nesting)
                false_nesting--;
        stream->top_if = top_if->next;
        __free_token(top_if);
        return 1;
}

static int handle_warning(struct stream *stream, struct token **line, struct tok
{
        warning(token->pos, "%s", show_token_sequence(token->next, 0));
        return 1;
}

static int handle_error(struct stream *stream, struct token **line, struct token
{
        sparse_error(token->pos, "%s", show_token_sequence(token->next, 0));
        return 1;
}

static int handle_nostdinc(struct stream *stream, struct token **line, struct to
{
        /*
         * Do we have any non-system includes?
         * Clear them out if so..
         */
        *sys_includepath = NULL;
        return 1;
}

static inline void update_inc_ptrs(const char ***where)
```



```
{

        if (*where <= dirafter_includepath) {
                dirafter_includepath++;
                /* If this was the entry that we prepend, don't
                 * rise the lower entries, even if they are at
                 * the same level. */
                if (where == &dirafter_includepath)
                        return;
        }
        if (*where <= sys_includepath) {
                sys_includepath++;
                if (where == &sys_includepath)
                        return;
        }
        if (*where <= isys_includepath) {
                isys_includepath++;
                if (where == &isys_includepath)
                        return;
        }

        /* angle_includepath is actually never updated, since we
         * don't suppport -iquote rught now. May change some day. */
        if (*where <= angle_includepath) {
                angle_includepath++;
                if (where == &angle_includepath)
                        return;
        }
}

/* Add a path before 'where' and update the pointers associated with the
 * includepath array */
static void add_path_entry(struct token *token, const char *path,
        const char ***where)
{
        const char **dst;
        const char *next;

        /* Need one free entry.. */
        if (includepath[INCLUDEPATHS-2])
                error_die(token->pos, "too many include path entries");

        /* check that this is not a duplicate */
        dst = includepath;
        while (*dst) {
                if (strcmp(*dst, path) == 0)
                        return;
                dst++;
        }
        next = path;
        dst = *where;

        update_inc_ptrs(where);

        /*
         * Move them all up starting at dst,
         * insert the new entry..
         */
        do {
                const char *tmp = *dst;
                *dst = next;
                next = tmp;
                dst++;
        } while (next);
}
```

```
static int handle_add_include(struct stream *stream, struct token **line, struct
{
        for (;;) {
                token = token->next;
                if (eof_token(token))
                        return 1;
                if (token_type(token) != TOKEN_STRING) {
                        warning(token->pos, "expected path string");
                        return 1;
                }
                add_path_entry(token, token->string->data, &isys_includepath);
        }
}

static int handle_add_isystem(struct stream *stream, struct token **line, struct
{
        for (;;) {
                token = token->next;
                if (eof_token(token))
                        return 1;
                if (token_type(token) != TOKEN_STRING) {
                        sparse_error(token->pos, "expected path string");
                        return 1;
                }
                add_path_entry(token, token->string->data, &sys_includepath);
        }
}

static int handle_add_system(struct stream *stream, struct token **line, struct
{
        for (;;) {
                token = token->next;
                if (eof_token(token))
                        return 1;
                if (token_type(token) != TOKEN_STRING) {
                        sparse_error(token->pos, "expected path string");
                        return 1;
                }
                add_path_entry(token, token->string->data, &dirafter_includepath
        }
}

/* Add to end on includepath list - no pointer updates */
static void add_dirafter_entry(struct token *token, const char *path)
{
        const char **dst = includepath;

        /* Need one free entry.. */
        if (includepath[INCLUDEPATHS-2])
                error_die(token->pos, "too many include path entries");

        /* Add to the end */
        while (*dst)
                dst++;
        *dst = path;
        dst++;
        *dst = NULL;
}

static int handle_add_dirafter(struct stream *stream, struct token **line, struc
{
        for (;;) {
                token = token->next;
                if (eof_token(token))
                        return 1;
                if (token_type(token) != TOKEN_STRING) {
```



```
                        sparse_error(token->pos, "expected path string");
                        return 1;
                }
                add_dirafter_entry(token, token->string->data);
        }
}

static int handle_split_include(struct stream *stream, struct token **line, stru
{
        /*
         * -I-
         *  From info gcc:
         *  Split the include path.  Any directories specified with '-I'
         *  options before '-I-' are searched only for headers requested with
         *  '#include "FILE"'; they are not searched for '#include <FILE>'.
         *  If additional directories are specified with '-I' options after
         *  the '-I-', those directories are searched for all '#include'
         *  directives.
         *  In addition, '-I-' inhibits the use of the directory of the current
         *  file directory as the first search directory for '#include "FILE"'.
         */
        quote_includepath = includepath+1;
        angle_includepath = sys_includepath;
        return 1;
}

/*
 * We replace "#pragma xxx" with "__pragma__" in the token
 * stream. Just as an example.
 *
 * We'll just #define that away for now, but the theory here
 * is that we can use this to insert arbitrary token sequences
 * to turn the pragmas into internal front-end sequences for
 * when we actually start caring about them.
 *
 * So eventually this will turn into some kind of extended
 * __attribute__() like thing, except called __pragma__(xxx).
 */
static int handle_pragma(struct stream *stream, struct token **line, struct toke
{
        struct token *next = *line;

        if (match_ident(token->next, &once_ident) && eof_token(token->next->next
                stream->once = 1;
                return 1;
        }
        token->ident = &pragma_ident;
        token->pos.newline = 1;
        token->pos.whitespace = 1;
        token->pos.pos = 1;
        *line = token;
        token->next = next;
        return 0;
}

/*
 * We ignore #line for now.
 */
static int handle_line(struct stream *stream, struct token **line, struct token
{
        return 1;
}

/*
 * Ignore "#ident".
 */
```

```
1975 static int handle_ident(struct stream *stream, struct token **line, struct token
1976 {
1977         return 1;
1978 }

1980 static int handle_nondirective(struct stream *stream, struct token **line, struc
1981 {
1982         sparse_error(token->pos, "unrecognized preprocessor line '%s'", show_tok
1983         return 1;
1984 }


1987 static void init_preprocessor(void)
1988 {
1989         int i;
1990         int stream = init_stream("preprocessor", -1, includepath);
1991         static struct {
1992                 const char *name;
1993                 int (*handler)(struct stream *, struct token **, struct token *)
1994         } normal[] = {
1995                 { "define",             handle_define },
1996                 { "weak_define",        handle_weak_define },
1997                 { "strong_define",      handle_strong_define },
1998                 { "undef",              handle_undef },
1999                 { "strong_undef",       handle_strong_undef },
2000                 { "warning",            handle_warning },
2001                 { "error",              handle_error },
2002                 { "include",            handle_include },
2003                 { "include_next",       handle_include_next },
2004                 { "pragma",             handle_pragma },
2005                 { "line",               handle_line },
2006                 { "ident",              handle_ident },

2008                 // our internal preprocessor tokens
2009                 { "nostdinc",      handle_nostdinc },
2010                 { "add_include",   handle_add_include },
2011                 { "add_isystem",   handle_add_isystem },
2012                 { "add_system",    handle_add_system },
2013                 { "add_dirafter",  handle_add_dirafter },
2014                 { "split_include", handle_split_include },
2015                 { "argv_include",  handle_argv_include },
2016         }, special[] = {
2017                 { "ifdef",      handle_ifdef },
2018                 { "ifndef",     handle_ifndef },
2019                 { "else",       handle_else },
2020                 { "endif",      handle_endif },
2021                 { "if",         handle_if },
2022                 { "elif",       handle_elif },
2023         };

2025         for (i = 0; i < ARRAY_SIZE(normal); i++) {
2026                 struct symbol *sym;
2027                 sym = create_symbol(stream, normal[i].name, SYM_PREPROCESSOR, NS
2028                 sym->handler = normal[i].handler;
2029                 sym->normal = 1;
2030         }
2031         for (i = 0; i < ARRAY_SIZE(special); i++) {
2032                 struct symbol *sym;
2033                 sym = create_symbol(stream, special[i].name, SYM_PREPROCESSOR, N
2034                 sym->handler = special[i].handler;
2035                 sym->normal = 0;
2036         }

2038         counter_macro = 0;
2039 }
```



```
2041 static void handle_preprocessor_line(struct stream *stream, struct token **line,
2042 {
2043         int (*handler)(struct stream *, struct token **, struct token *);
2044         struct token *token = start->next;
2045         int is_normal = 1;

2047         if (eof_token(token))
2048                 return;

2050         if (token_type(token) == TOKEN_IDENT) {
2051                 struct symbol *sym = lookup_symbol(token->ident, NS_PREPROCESSOR
2052                 if (sym) {
2053                         handler = sym->handler;
2054                         is_normal = sym->normal;
2055                 } else {
2056                         handler = handle_nondirective;
2057                 }
2058         } else if (token_type(token) == TOKEN_NUMBER) {
2059                 handler = handle_line;
2060         } else {
2061                 handler = handle_nondirective;
2062         }

2064         if (is_normal) {
2065                 dirty_stream(stream);
2066                 if (false_nesting)
2067                         goto out;
2068         }
2069         if (!handler(stream, line, token))      /* all set */
2070                 return;

2072 out:
2073         free_preprocessor_line(token);
2074 }

2076 static void preprocessor_line(struct stream *stream, struct token **line)
2077 {
2078         struct token *start = *line, *next;
2079         struct token **tp = &start->next;

2081         for (;;) {
2082                 next = *tp;
2083                 if (next->pos.newline)
2084                         break;
2085                 tp = &next->next;
2086         }
2087         *line = next;
2088         *tp = &eof_token_entry;
2089         handle_preprocessor_line(stream, line, start);
2090 }

2092 static void do_preprocess(struct token **list)
2093 {
2094         struct token *next;

2096         while (!eof_token(next = scan_next(list))) {
2097                 struct stream *stream = input_streams + next->pos.stream;

2099                 if (next->pos.newline && match_op(next, '#')) {
2100                         if (!next->pos.noexpand) {
2101                                 preprocessor_line(stream, list);
2102                                 __free_token(next);     /* Free the '#' token */
2103                                 continue;
2104                         }
2105                 }
```

```
2107                 switch (token_type(next)) {
2108                 case TOKEN_STREAMEND:
2109                         if (stream->top_if) {
2110                                 nesting_error(stream);
2111                                 sparse_error(stream->top_if->pos, "unterminated
2112                                 stream->top_if = NULL;
2113                                 false_nesting = 0;
2114                         }
2115                         if (!stream->dirty)
2116                                 stream->constant = CONSTANT_FILE_YES;
2117                         *list = next->next;
2118                         continue;
2119                 case TOKEN_STREAMBEGIN:
2120                         *list = next->next;
2121                         continue;

2123                 default:
2124                         dirty_stream(stream);
2125                         if (false_nesting) {
2126                                 *list = next->next;
2127                                 __free_token(next);
2128                                 continue;
2129                         }

2131                         if (token_type(next) != TOKEN_IDENT ||
2132                             expand_one_symbol(list))
2133                                 list = &next->next;
2134                 }
2135         }
2136 }

2138 void init_include_path(void)
2139 {
2140         FILE *fp;
2141         char path[256];
2142         char arch[32];
2143         char os[32];

2145         fp = popen("/bin/uname -m", "r");
2146         if (!fp)
2147                 return;
2148         if (!fgets(arch, sizeof(arch) - 1, fp))
2149                 return;
2150         pclose(fp);
2151         if (arch[strlen(arch) - 1] == '\n')
2152                 arch[strlen(arch) - 1] = '\0';

2154         fp = popen("/bin/uname -o", "r");
2155         if (!fp)
2156                 return;
2157         fgets(os, sizeof(os) - 1, fp);
2158         pclose(fp);

2160         if (strcmp(os, "GNU/Linux\n") != 0)
2161                 return;
2162         strcpy(os, "linux-gnu");

2164         snprintf(path, sizeof(path), "/usr/include/%s-%s/", arch, os);
2165         add_pre_buffer("#add_system \"%s/\"\n", path);
2166 }

2168 struct token * preprocess(struct token *token)
2169 {
2170         preprocessing = 1;
2171         init_preprocessor();
2172         do_preprocess(&token);
```



```
2174         // Drop all expressions from preprocessing, they're not used any more.
2175         // This is not true when we have multiple files, though ;/
2176         // clear_expression_alloc();
2177         preprocessing = 0;

2179         return token;
2180 }

2182 static void dump_macro(struct symbol *sym)
2183 {
2184         int nargs = sym->arglist ? sym->arglist->count.normal : 0;
2185         struct token *args[nargs];
2186         struct token *token;

2188         printf("#define %s", show_ident(sym->ident));
2189         token = sym->arglist;
2190         if (token) {
2191                 const char *sep = "";
2192                 int narg = 0;
2193                 putchar('(');
2194                 for (; !eof_token(token); token = token->next) {
2195                         if (token_type(token) == TOKEN_ARG_COUNT)
2196                                 continue;
2197                         printf("%s%s", sep, show_token(token));
2198                         args[narg++] = token;
2199                         sep = ", ";
2200                 }
2201                 putchar(')');
2202         }
2203         putchar(' ');

2205         token = sym->expansion;
2206         while (!eof_token(token)) {
2207                 struct token *next = token->next;
2208                 switch (token_type(token)) {
2209                 case TOKEN_UNTAINT:
2210                         break;
2211                 case TOKEN_MACRO_ARGUMENT:
2212                         token = args[token->argnum];
2213                         /* fall-through */
2214                 default:
2215                         printf("%s", show_token(token));
2216                         if (next->pos.whitespace)
2217                                 putchar(' ');
2218                 }
2219                 token = next;
2220         }
2221         putchar('\n');
2222 }

2224 void dump_macro_definitions(void)
2225 {
2226         struct ident *name;

2228         FOR_EACH_PTR(macros, name) {
2229                 struct symbol *sym = lookup_macro(name);
2230                 if (sym)
2231                         dump_macro(sym);
2232         } END_FOR_EACH_PTR(name);
2233 }
```

```
**********************************************************
    5075 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/ptrlist.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * ptrlist.c
 *
 * Pointer list manipulation
 *
 * (C) Copyright Linus Torvalds 2003-2005
 */
#include <stdlib.h>
#include <string.h>
#include <assert.h>

#include "ptrlist.h"
#include "allocate.h"
#include "compat.h"

__DECLARE_ALLOCATOR(struct ptr_list, ptrlist);
__ALLOCATOR(struct ptr_list, "ptr list", ptrlist);
__ALLOCATOR(struct ptr_list, "rl ptr list", rl_ptrlist);

int ptr_list_size(struct ptr_list *head)
{
        int nr = 0;

        if (head) {
                struct ptr_list *list = head;
                do {
                        nr += list->nr - list->rm;
                } while ((list = list->next) != head);
        }
        return nr;
}

/*
 * Linearize the entries of a list up to a total of 'max',
 * and return the nr of entries linearized.
 *
 * The array to linearize into (second argument) should really
 * be "void *x[]", but we want to let people fill in any kind
 * of pointer array, so let's just call it "void **".
 */
int linearize_ptr_list(struct ptr_list *head, void **arr, int max)
{
        int nr = 0;
        if (head && max > 0) {
                struct ptr_list *list = head;

                do {
                        int i = list->nr;
                        if (i > max)
                                i = max;
                        memcpy(arr, list->list, i*sizeof(void *));
                        arr += i;
                        nr += i;
                        max -= i;
                        if (!max)
                                break;
                } while ((list = list->next) != head);
        }
        return nr;
}
```



```
/*
 * When we've walked the list and deleted entries,
 * we may need to re-pack it so that we don't have
 * any empty blocks left (empty blocks upset the
 * walking code
 */
void pack_ptr_list(struct ptr_list **listp)
{
        struct ptr_list *head = *listp;

        if (head) {
                struct ptr_list *entry = head;
                do {
                        struct ptr_list *next;
restart:
                        next = entry->next;
                        if (!entry->nr) {
                                struct ptr_list *prev;
                                if (next == entry) {
                                        __free_ptrlist(entry);
                                        *listp = NULL;
                                        return;
                                }
                                prev = entry->prev;
                                prev->next = next;
                                next->prev = prev;
                                __free_ptrlist(entry);
                                if (entry == head) {
                                        *listp = next;
                                        head = next;
                                        entry = next;
                                        goto restart;
                                }
                        }
                        entry = next;
                } while (entry != head);
        }
}

void split_ptr_list_head(struct ptr_list *head)
{
        int old = head->nr, nr = old / 2;
        struct ptr_list *newlist = __alloc_ptrlist(0);
        struct ptr_list *next = head->next;

        old -= nr;
        head->nr = old;
        newlist->next = next;
        next->prev = newlist;
        newlist->prev = head;
        head->next = newlist;
        newlist->nr = nr;
        memcpy(newlist->list, head->list + old, nr * sizeof(void *));
        memset(head->list + old, 0xf0, nr * sizeof(void *));
}

int rl_ptrlist_hack;
void **__add_ptr_list(struct ptr_list **listp, void *ptr, unsigned long tag)
{
        struct ptr_list *list = *listp;
        struct ptr_list *last = NULL; /* gcc complains needlessly */
        void **ret;
        int nr;

        /* The low two bits are reserved for tags */
```

```
127         assert((3 & (unsigned long)ptr) == 0);
128         assert((~3 & tag) == 0);
129         ptr = (void *)(tag | (unsigned long)ptr);

131         if (!list || (nr = (last = list->prev)->nr) >= LIST_NODE_NR) {
132                 struct ptr_list *newlist;

134                 if (rl_ptrlist_hack)
135                         newlist = __alloc_rl_ptrlist(0);
136                 else
137                         newlist = __alloc_ptrlist(0);
138                 if (!list) {
139                         newlist->next = newlist;
140                         newlist->prev = newlist;
141                         *listp = newlist;
142                 } else {
143                         newlist->prev = last;
144                         newlist->next = list;
145                         list->prev = newlist;
146                         last->next = newlist;
147                 }
148                 last = newlist;
149                 nr = 0;
150         }
151         ret = last->list + nr;
152         *ret = ptr;
153         nr++;
154         last->nr = nr;
155         return ret;
156 }

158 int delete_ptr_list_entry(struct ptr_list **list, void *entry, int count)
159 {
160         void *ptr;

162         FOR_EACH_PTR(*list, ptr) {
163                 if (ptr == entry) {
164                         DELETE_CURRENT_PTR(ptr);
165                         if (!--count)
166                                 goto out;
167                 }
168         } END_FOR_EACH_PTR(ptr);
169         assert(count <= 0);
170 out:
171         pack_ptr_list(list);
172         return count;
173 }

175 int replace_ptr_list_entry(struct ptr_list **list, void *old_ptr, void *new_ptr,
176 {
177         void *ptr;

179         FOR_EACH_PTR(*list, ptr) {
180                 if (ptr==old_ptr) {
181                         REPLACE_CURRENT_PTR(ptr, new_ptr);
182                         if (!--count)
183                                 goto out;
184                 }
185         }END_FOR_EACH_PTR(ptr);
186         assert(count <= 0);
187 out:
188         return count;
189 }

191 /* This removes the last entry, but doesn't pack the ptr list */
192 void * undo_ptr_list_last(struct ptr_list **head)
```



```
193 {
194         struct ptr_list *last, *first = *head;

196         if (!first)
197                 return NULL;
198         last = first;
199         do {
200                 last = last->prev;
201                 if (last->nr) {
202                         void *ptr;
203                         int nr = --last->nr;
204                         ptr = last->list[nr];
205                         last->list[nr] = (void *)0xf1f1f1f1;
206                         return ptr;
207                 }
208         } while (last != first);
209         return NULL;
210 }

212 void * delete_ptr_list_last(struct ptr_list **head)
213 {
214         void *ptr = NULL;
215         struct ptr_list *last, *first = *head;

217         if (!first)
218                 return NULL;
219         last = first->prev;
220         if (last->nr)
221                 ptr = last->list[--last->nr];
222         if (last->nr <=0) {
223                 first->prev = last->prev;
224                 last->prev->next = first;
225                 if (last == first)
226                         *head = NULL;
227                 __free_ptrlist(last);
228         }
229         return ptr;
230 }

232 void concat_ptr_list(struct ptr_list *a, struct ptr_list **b)
233 {
234         void *entry;
235         FOR_EACH_PTR(a, entry) {
236                 add_ptr_list(b, entry);
237         } END_FOR_EACH_PTR(entry);
238 }

240 void __free_ptr_list(struct ptr_list **listp)
241 {
242         struct ptr_list *tmp, *list = *listp;

244         if (!list)
245                 return;

247         list->prev->next = NULL;
248         while (list) {
249                 tmp = list;
250                 list = list->next;
251                 __free_ptrlist(tmp);
252         }

254         *listp = NULL;
255 }
```

```
**********************************************************
    9611 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/ptrlist.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef PTR_LIST_H
#define PTR_LIST_H

#include <stdlib.h>

/*
 * Generic pointer list manipulation code.
 *
 * (C) Copyright Linus Torvalds 2003-2005
 */

/* Silly type-safety check ;) */
#define DECLARE_PTR_LIST(listname,type) struct listname { type *list[1]; }
#define CHECK_TYPE(head,ptr)            (void)(&(ptr) == &(head)->list[0])
#define TYPEOF(head)                    __typeof__(&(head)->list[0])
#define VRFY_PTR_LIST(head)             (void)(sizeof((head)->list[0]))

/*
 * The "unnecessary" statement expression is there to shut up a totally
 * bogus gcc warning about unused expressions, brought on by the fact
 * that we cast the result to the proper type.
 */
#define MKTYPE(head,expr)               ({ (TYPEOF(head))(expr); })

#define LIST_NODE_NR (29)

struct ptr_list {
        int nr:8;
        int rm:8;
        struct ptr_list *prev;
        struct ptr_list *next;
        void *list[LIST_NODE_NR];
};

#define ptr_list_empty(x) ((x) == NULL)

void * undo_ptr_list_last(struct ptr_list **head);
void * delete_ptr_list_last(struct ptr_list **head);
int delete_ptr_list_entry(struct ptr_list **, void *, int);
int replace_ptr_list_entry(struct ptr_list **, void *old, void *new, int);
extern void sort_list(struct ptr_list **, int (*)(const void *, const void *));

extern void **__add_ptr_list(struct ptr_list **, void *, unsigned long);
extern void concat_ptr_list(struct ptr_list *a, struct ptr_list **b);
extern void __free_ptr_list(struct ptr_list **);
extern int ptr_list_size(struct ptr_list *);
extern int linearize_ptr_list(struct ptr_list *, void **, int);

/*
 * Hey, who said that you can't do overloading in C?
 *
 * You just have to be creative, and use some gcc
 * extensions..
 */
#define add_ptr_list_tag(list,entry,tag) \
        MKTYPE(*(list), (CHECK_TYPE(*(list),(entry)),__add_ptr_list((struct ptr_
#define add_ptr_list_notag(list,entry)
        MKTYPE(*(list), (CHECK_TYPE(*(list),(entry)),__add_ptr_list((struct ptr_
                                                                    (void *)((un
                                                                    (unsigned lo
```



```
#define add_ptr_list(list,entry) \
        add_ptr_list_tag(list,entry,0)
#define free_ptr_list(list) \
        do { VRFY_PTR_LIST(*(list)); __free_ptr_list((struct ptr_list **)(list))

#define PTR_ENTRY_NOTAG(h,i)    ((h)->list[i])
#define PTR_ENTRY(h,i)  (void *)(~3UL & (unsigned long)PTR_ENTRY_NOTAG(h,i))

static inline void *first_ptr_list(struct ptr_list *list)
{
        struct ptr_list *head = list;

        if (!list)
                return NULL;

        while (list->nr == 0) {
                list = list->next;
                if (list == head)
                        return NULL;
        }
        return PTR_ENTRY(list, 0);
}

static inline void *last_ptr_list(struct ptr_list *list)
{
        struct ptr_list *head = list;

        if (!list)
                return NULL;
        list = list->prev;
        while (list->nr == 0) {
                if (list == head)
                        return NULL;
                list = list->prev;
        }
        return PTR_ENTRY(list, list->nr-1);
}

#define PTR_DEREF(__head, idx, PTR_ENTRY) ({
        struct ptr_list *__list = __head;
        while (__list && __list->nr == 0) {
                __list = __list->next;
                if (__list == __head)
                        __list = NULL;
        }
        __list ? PTR_ENTRY(__list, idx) : NULL;
})

#define DO_PREPARE(head, ptr, __head, __list, __nr, PTR_ENTRY)
        do {
                struct ptr_list *__head = (struct ptr_list *) (head);
                struct ptr_list *__list = __head;
                int __nr = 0;
                CHECK_TYPE(head,ptr);
                ptr = PTR_DEREF(__head, 0, PTR_ENTRY);

#define DO_NEXT(ptr, __head, __list, __nr, PTR_ENTRY)
                if (ptr) {
                        if (++__nr < __list->nr) {
                                ptr = PTR_ENTRY(__list,__nr);
                        } else {
                                __list = __list->next;
                                ptr = NULL;
                                while (__list->nr == 0 && __list != __head)
                                        __list = __list->next;
                                if (__list != __head) {
```

```
127                                     __nr = 0;
128                                     ptr = PTR_ENTRY(__list,0);
129                             }
130                     }
131             }

133 #define DO_RESET(ptr, __head, __list, __nr, PTR_ENTRY)
134         do {
135                 __nr = 0;
136                 __list = __head;
137                 if (__head) ptr = PTR_DEREF(__head, 0, PTR_ENTRY);
138         } while (0)

140 #define DO_FINISH(ptr, __head, __list, __nr)
141                 (void)(__nr); /* Sanity-check nesting */
142         } while (0)

144 #define PREPARE_PTR_LIST(head, ptr) \
145         DO_PREPARE(head, ptr, __head##ptr, __list##ptr, __nr##ptr, PTR_ENTRY)

147 #define NEXT_PTR_LIST(ptr) \
148         DO_NEXT(ptr, __head##ptr, __list##ptr, __nr##ptr, PTR_ENTRY)

150 #define RESET_PTR_LIST(ptr) \
151         DO_RESET(ptr, __head##ptr, __list##ptr, __nr##ptr, PTR_ENTRY)

153 #define FINISH_PTR_LIST(ptr) \
154         DO_FINISH(ptr, __head##ptr, __list##ptr, __nr##ptr)

156 #define DO_FOR_EACH(head, ptr, __head, __list, __nr, PTR_ENTRY) do {
157         struct ptr_list *__head = (struct ptr_list *) (head);
158         struct ptr_list *__list = __head;
159         CHECK_TYPE(head,ptr);
160         if (__head) {
161                 do { int __nr;
162                         for (__nr = 0; __nr < __list->nr; __nr++) {
163                                 do {
164                                         ptr = PTR_ENTRY(__list,__nr);
165                                         if (__list->rm && !ptr)
166                                                 continue;
167                                         do {

169 #define DO_END_FOR_EACH(ptr, __head, __list, __nr)
170                                         } while (0);
171                                 } while (0);
172                         }
173                 } while ((__list = __list->next) != __head);
174         }
175 } while (0)

177 #define DO_FOR_EACH_REVERSE(head, ptr, __head, __list, __nr, PTR_ENTRY) do {
178         struct ptr_list *__head = (struct ptr_list *) (head);
179         struct ptr_list *__list = __head;
180         CHECK_TYPE(head,ptr);
181         if (__head) {
182                 do { int __nr;
183                         __list = __list->prev;
184                         __nr = __list->nr;
185                         while (--__nr >= 0) {
186                                 do {
187                                         ptr = PTR_ENTRY(__list,__nr);
188                                         if (__list->rm && !ptr)
189                                                 continue;
190                                         do {
```



```
193 #define DO_END_FOR_EACH_REVERSE(ptr, __head, __list, __nr)
194                                         } while (0);
195                                 } while (0);
196                         }
197                 } while (__list != __head);
198         }
199 } while (0)

201 #define DO_REVERSE(ptr, __head, __list, __nr, new, __newhead,
202                    __newlist, __newnr, PTR_ENTRY) do {
203         struct ptr_list *__newhead = __head;
204         struct ptr_list *__newlist = __list;
205         int __newnr = __nr;
206         new = ptr;
207         goto __inside##new;
208         if (1) {
209                 do {
210                         __newlist = __newlist->prev;
211                         __newnr = __newlist->nr;
212         __inside##new:
213                         while (--__newnr >= 0) {
214                                 do {
215                                         new = PTR_ENTRY(__newlist,__newnr);
216                                         do {

218 #define RECURSE_PTR_REVERSE(ptr, new)
219         DO_REVERSE(ptr, __head##ptr, __list##ptr, __nr##ptr,
220                    new, __head##new, __list##new, __nr##new, PTR_ENTRY)

222 #define DO_THIS_ADDRESS(ptr, __head, __list, __nr)
223         ((__typeof__(&(ptr))) (__list->list + __nr))

225 #define FOR_EACH_PTR(head, ptr) \
226         DO_FOR_EACH(head, ptr, __head##ptr, __list##ptr, __nr##ptr, PTR_ENTRY)

228 #define END_FOR_EACH_PTR(ptr) \
229         DO_END_FOR_EACH(ptr, __head##ptr, __list##ptr, __nr##ptr)

231 #define FOR_EACH_PTR_NOTAG(head, ptr) \
232         DO_FOR_EACH(head, ptr, __head##ptr, __list##ptr, __nr##ptr, PTR_ENTRY_NO

234 #define END_FOR_EACH_PTR_NOTAG(ptr) END_FOR_EACH_PTR(ptr)

236 #define FOR_EACH_PTR_REVERSE(head, ptr) \
237         DO_FOR_EACH_REVERSE(head, ptr, __head##ptr, __list##ptr, __nr##ptr, PTR_

239 #define END_FOR_EACH_PTR_REVERSE(ptr) \
240         DO_END_FOR_EACH_REVERSE(ptr, __head##ptr, __list##ptr, __nr##ptr)

242 #define FOR_EACH_PTR_REVERSE_NOTAG(head, ptr) \
243         DO_FOR_EACH_REVERSE(head, ptr, __head##ptr, __list##ptr, __nr##ptr, PTR_

245 #define END_FOR_EACH_PTR_REVERSE_NOTAG(ptr) END_FOR_EACH_PTR_REVERSE(ptr)

247 #define THIS_ADDRESS(ptr) \
248         DO_THIS_ADDRESS(ptr, __head##ptr, __list##ptr, __nr##ptr)

250 extern void split_ptr_list_head(struct ptr_list *);

252 #define DO_SPLIT(ptr, __head, __list, __nr) do {
253         split_ptr_list_head(__list);
254         if (__nr >= __list->nr) {
255                 __nr -= __list->nr;
256                 __list = __list->next;
257         };
258 } while (0)
```

```
#define DO_INSERT_CURRENT(new, ptr, __head, __list, __nr) do {
        void **__this, **__last;
        if (__list->nr == LIST_NODE_NR)
                DO_SPLIT(ptr, __head, __list, __nr);
        __this = __list->list + __nr;
        __last = __list->list + __list->nr - 1;
        while (__last >= __this) {
                __last[1] = __last[0];
                __last--;
        }
        *__this = (new);
        __list->nr++;
} while (0)

#define INSERT_CURRENT(new, ptr) \
        DO_INSERT_CURRENT(new, ptr, __head##ptr, __list##ptr, __nr##ptr)

#define DO_DELETE_CURRENT(ptr, __head, __list, __nr) do {
        void **__this = __list->list + __nr;
        void **__last = __list->list + __list->nr - 1;
        while (__this < __last) {
                __this[0] = __this[1];
                __this++;
        }
        *__this = (void *)0xf0f0f0f0;
        __list->nr--; __nr--;
} while (0)

#define DELETE_CURRENT_PTR(ptr) \
        DO_DELETE_CURRENT(ptr, __head##ptr, __list##ptr, __nr##ptr)

#define REPLACE_CURRENT_PTR(ptr, new_ptr)
        do { *THIS_ADDRESS(ptr) = (new_ptr); } while (0)

#define DO_MARK_CURRENT_DELETED(ptr, __list) do {       \
                REPLACE_CURRENT_PTR(ptr, NULL);         \
                __list->rm++;                           \
        } while (0)

#define MARK_CURRENT_DELETED(ptr) \
        DO_MARK_CURRENT_DELETED(ptr, __list##ptr)

extern void pack_ptr_list(struct ptr_list **);

#define PACK_PTR_LIST(x) pack_ptr_list((struct ptr_list **)(x))

static inline void update_tag(void *p, unsigned long tag)
{
        unsigned long *ptr = p;
        *ptr = tag | (~3UL & *ptr);
}

static inline void *tag_ptr(void *ptr, unsigned long tag)
{
        return (void *)(tag | (unsigned long)ptr);
}

#define CURRENT_TAG(ptr) (3 & (unsigned long)*THIS_ADDRESS(ptr))
#define TAG_CURRENT(ptr,val)    update_tag(THIS_ADDRESS(ptr),val)

#endif /* PTR_LIST_H */
```

```
**********************************************************
    3667 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/scope.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Symbol scoping.
 *
 * This is pretty trivial.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "lib.h"
#include "allocate.h"
#include "symbol.h"
#include "scope.h"

static struct scope builtin_scope = { .next = &builtin_scope };

struct scope    *block_scope = &builtin_scope,          // regular automatic var
                *function_scope = &builtin_scope,       // labels, arguments etc
                *file_scope = &builtin_scope,           // static
                *global_scope = &builtin_scope;         // externally visible

void bind_scope(struct symbol *sym, struct scope *scope)
{
        sym->scope = scope;
        add_symbol(&scope->symbols, sym);
}

void rebind_scope(struct symbol *sym, struct scope *new)
{
        struct scope *old = sym->scope;

        if (old == new)
                return;

        if (old)
                delete_ptr_list_entry((struct ptr_list**) &old->symbols, sym, 1)

        bind_scope(sym, new);
}
```



```
static void start_scope(struct scope **s, struct position pos)
{
        struct scope *scope = __alloc_scope(0);
        memset(scope, 0, sizeof(*scope));
        scope->token = __alloc_token(0);
        scope->token->pos = pos;
        scope->next = *s;
        *s = scope;
}

void start_file_scope(void)
{
        struct scope *scope = __alloc_scope(0);

        memset(scope, 0, sizeof(*scope));
        scope->next = &builtin_scope;
        file_scope = scope;

        /* top-level stuff defaults to file scope, "extern" etc will choose glob
        function_scope = scope;
        block_scope = scope;
}

void start_symbol_scope(struct position pos)
{
        start_scope(&block_scope, pos);
}

void start_function_scope(struct position pos)
{
        start_scope(&function_scope, pos);
        start_scope(&block_scope, pos);
}

static void remove_symbol_scope(struct symbol *sym)
{
        struct symbol **ptr = &sym->ident->symbols;

        while (*ptr != sym)
                ptr = &(*ptr)->next_id;
        *ptr = sym->next_id;
}

static void end_scope(struct scope **s)
{
        struct scope *scope = *s;
        struct symbol_list *symbols = scope->symbols;
        struct symbol *sym;

        *s = scope->next;
        scope->symbols = NULL;
        FOR_EACH_PTR(symbols, sym) {
                remove_symbol_scope(sym);
        } END_FOR_EACH_PTR(sym);
}

void end_file_scope(void)
{
        end_scope(&file_scope);
}

void new_file_scope(void)
{
        if (file_scope != &builtin_scope)
                end_file_scope();
```

```
127         start_file_scope();
128 }

130 void end_symbol_scope(void)
131 {
132         end_scope(&block_scope);
133 }

135 void end_function_scope(void)
136 {
137         end_scope(&block_scope);
138         end_scope(&function_scope);
139 }

141 int is_outer_scope(struct scope *scope)
142 {
143         if (scope == block_scope)
144                 return 0;
145         if (scope == &builtin_scope && block_scope->next == &builtin_scope)
146                 return 0;
147         return 1;
148 }
```

```
**********************************************************
    2072 Fri Dec 21 15:00:14 2018
new/usr/src/tools/smatch/src/scope.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef SCOPE_H
#define SCOPE_H
/*
 * Symbol scoping is pretty simple.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

struct symbol;
struct position;

struct scope {
        struct token *token;            /* Scope start information */
        struct symbol_list *symbols;    /* List of symbols in this scope */
        struct scope *next;
};

extern struct scope
                *block_scope,
                *function_scope,
                *file_scope,
                *global_scope;

static inline int toplevel(struct scope *scope)
{
        return scope == file_scope || scope == global_scope;
}

extern void start_file_scope(void);
extern void end_file_scope(void);
extern void new_file_scope(void);

extern void start_symbol_scope(struct position pos);
extern void end_symbol_scope(void);

extern void start_function_scope(struct position pos);
extern void end_function_scope(void);

extern void bind_scope(struct symbol *, struct scope *);
extern void rebind_scope(struct symbol *, struct scope *);
```



```
extern int is_outer_scope(struct scope *);
#endif
```

```
**********************************************************
   28032 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/show-parse.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/show-parse.c
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 *
 * Print out results of parsing for debugging and testing.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "scope.h"
#include "expression.h"
#include "target.h"

static int show_symbol_expr(struct symbol *sym);
static int show_string_expr(struct expression *expr);

static void do_debug_symbol(struct symbol *sym, int indent)
{
        static const char indent_string[] = "                                                "
        static const char *typestr[] = {
                [SYM_UNINITIALIZED] = "none",
                [SYM_PREPROCESSOR] = "cpp.",
                [SYM_BASETYPE] = "base",
                [SYM_NODE] = "node",
                [SYM_PTR] = "ptr.",
                [SYM_FN] = "fn..",
                [SYM_ARRAY] = "arry",
                [SYM_STRUCT] = "strt",
                [SYM_UNION] = "unin",
                [SYM_ENUM] = "enum",
```



```
                [SYM_TYPEDEF] = "tdef",
                [SYM_TYPEOF] = "tpof",
                [SYM_MEMBER] = "memb",
                [SYM_BITFIELD] = "bitf",
                [SYM_LABEL] = "labl",
                [SYM_RESTRICT] = "rstr",
                [SYM_FOULED] = "foul",
                [SYM_BAD] = "bad.",
        };
        struct context *context;
        int i;

        if (!sym)
                return;
        fprintf(stderr, "%.*s%s%3d:%lu %s %s (as: %d) %p (%s:%d:%d) %s\n",
                indent, indent_string, typestr[sym->type],
                sym->bit_size, sym->ctype.alignment,
                modifier_string(sym->ctype.modifiers), show_ident(sym->ident), s
                sym, stream_name(sym->pos.stream), sym->pos.line, sym->pos.pos,
                builtin_typename(sym) ?: "");
        i = 0;
        FOR_EACH_PTR(sym->ctype.contexts, context) {
                /* FIXME: should print context expression */
                fprintf(stderr, "< context%d: in=%d, out=%d\n",
                        i, context->in, context->out);
                fprintf(stderr, "  end context%d >\n", i);
                i++;
        } END_FOR_EACH_PTR(context);
        if (sym->type == SYM_FN) {
                struct symbol *arg;
                i = 0;
                FOR_EACH_PTR(sym->arguments, arg) {
                        fprintf(stderr, "< arg%d:\n", i);
                        do_debug_symbol(arg, 0);
                        fprintf(stderr, "  end arg%d >\n", i);
                        i++;
                } END_FOR_EACH_PTR(arg);
        }
        do_debug_symbol(sym->ctype.base_type, indent+2);
}

void debug_symbol(struct symbol *sym)
{
        do_debug_symbol(sym, 0);
}

/*
 * Symbol type printout. The type system is by far the most
 * complicated part of C - everything else is trivial.
 */
const char *modifier_string(unsigned long mod)
{
        static char buffer[100];
        int len = 0;
        int i;
        struct mod_name {
                unsigned long mod;
                const char *name;
        } *m;

        static struct mod_name mod_names[] = {
                {MOD_AUTO,              "auto"},
                {MOD_REGISTER,          "register"},
                {MOD_STATIC,            "static"},
                {MOD_EXTERN,            "extern"},
                {MOD_CONST,             "const"},
```

```
127                 {MOD_VOLATILE,          "volatile"},
128                 {MOD_SIGNED,            "[signed]"},
129                 {MOD_UNSIGNED,          "[unsigned]"},
130                 {MOD_CHAR,              "[char]"},
131                 {MOD_SHORT,             "[short]"},
132                 {MOD_LONG,              "[long]"},
133                 {MOD_LONGLONG,          "[long long]"},
134                 {MOD_LONGLONGLONG,      "[long long long]"},
135                 {MOD_TYPEDEF,           "[typedef]"},
136                 {MOD_TLS,               "[tls]"},
137                 {MOD_INLINE,            "inline"},
138                 {MOD_ADDRESSABLE,       "[addressable]"},
139                 {MOD_NOCAST,            "[nocast]"},
140                 {MOD_NODEREF,           "[noderef]"},
141                 {MOD_ACCESSED,          "[accessed]"},
142                 {MOD_TOPLEVEL,          "[toplevel]"},
143                 {MOD_ASSIGNED,          "[assigned]"},
144                 {MOD_TYPE,              "[type]"},
145                 {MOD_SAFE,              "[safe]"},
146                 {MOD_USERTYPE,          "[usertype]"},
147                 {MOD_NORETURN,          "[noreturn]"},
148                 {MOD_EXPLICITLY_SIGNED, "[explicitly-signed]"},
149                 {MOD_BITWISE,           "[bitwise]"},
150                 {MOD_PURE,              "[pure]"},
151         };

153         for (i = 0; i < ARRAY_SIZE(mod_names); i++) {
154                 m = mod_names + i;
155                 if (mod & m->mod) {
156                         char c;
157                         const char *name = m->name;
158                         while ((c = *name++) != '\0' && len + 2 < sizeof buffer)
159                                 buffer[len++] = c;
160                         buffer[len++] = ' ';
161                 }
162         }
163         buffer[len] = 0;
164         return buffer;
165 }

167 static void show_struct_member(struct symbol *sym)
168 {
169         printf("\t%s:%d:%ld at offset %ld.%d", show_ident(sym->ident), sym->bit_
170         printf("\n");
171 }

173 void show_symbol_list(struct symbol_list *list, const char *sep)
174 {
175         struct symbol *sym;
176         const char *prepend = "";

178         FOR_EACH_PTR(list, sym) {
179                 puts(prepend);
180                 prepend = ", ";
181                 show_symbol(sym);
182         } END_FOR_EACH_PTR(sym);
183 }

185 struct type_name {
186         char *start;
187         char *end;
188 };

190 static void FORMAT_ATTR(2) prepend(struct type_name *name, const char *fmt, ...)
191 {
192         static char buffer[512];
```



```
193         int n;

195         va_list args;
196         va_start(args, fmt);
197         n = vsprintf(buffer, fmt, args);
198         va_end(args);

200         name->start -= n;
201         memcpy(name->start, buffer, n);
202 }

204 static void FORMAT_ATTR(2) append(struct type_name *name, const char *fmt, ...)
205 {
206         static char buffer[512];
207         int n;

209         va_list args;
210         va_start(args, fmt);
211         n = vsprintf(buffer, fmt, args);
212         va_end(args);

214         memcpy(name->end, buffer, n);
215         name->end += n;
216 }

218 static struct ctype_name {
219         struct symbol *sym;
220         const char *name;
221 } typenames[] = {
222         { & char_ctype,  "char" },
223         { &schar_ctype,  "signed char" },
224         { &uchar_ctype,  "unsigned char" },
225         { & short_ctype, "short" },
226         { &sshort_ctype, "signed short" },
227         { &ushort_ctype, "unsigned short" },
228         { & int_ctype,   "int" },
229         { &sint_ctype,   "signed int" },
230         { &uint_ctype,   "unsigned int" },
231         { &slong_ctype,  "signed long" },
232         { & long_ctype,  "long" },
233         { &ulong_ctype,  "unsigned long" },
234         { & llong_ctype, "long long" },
235         { &sllong_ctype, "signed long long" },
236         { &ullong_ctype, "unsigned long long" },
237         { & lllong_ctype, "long long long" },
238         { &slllong_ctype, "signed long long long" },
239         { &ulllong_ctype, "unsigned long long long" },

241         { &void_ctype,   "void" },
242         { &bool_ctype,   "bool" },
243         { &string_ctype, "string" },

245         { &float_ctype,  "float" },
246         { &double_ctype, "double" },
247         { &ldouble_ctype,"long double" },
248         { &incomplete_ctype, "incomplete type" },
249         { &int_type, "abstract int" },
250         { &fp_type, "abstract fp" },
251         { &label_ctype, "label type" },
252         { &bad_ctype, "bad type" },
253 };

255 const char *builtin_typename(struct symbol *sym)
256 {
257         int i;
```

```
259         for (i = 0; i < ARRAY_SIZE(typenames); i++)
260                 if (typenames[i].sym == sym)
261                         return typenames[i].name;
262         return NULL;
263 }

265 const char *builtin_ctypename(struct ctype *ctype)
266 {
267         int i;

269         for (i = 0; i < ARRAY_SIZE(typenames); i++)
270                 if (&typenames[i].sym->ctype == ctype)
271                         return typenames[i].name;
272         return NULL;
273 }

275 static void do_show_type(struct symbol *sym, struct type_name *name)
276 {
277         const char *typename;
278         unsigned long mod = 0;
279         int as = 0;
280         int was_ptr = 0;
281         int restr = 0;
282         int fouled = 0;

284 deeper:
285         if (!sym || (sym->type != SYM_NODE && sym->type != SYM_ARRAY &&
286                      sym->type != SYM_BITFIELD)) {
287                 const char *s;
288                 size_t len;

290                 if (as)
291                         prepend(name, "<asn:%d>", as);

293                 s = modifier_string(mod);
294                 len = strlen(s);
295                 name->start -= len;
296                 memcpy(name->start, s, len);
297                 mod = 0;
298                 as = 0;
299         }

301         if (!sym)
302                 goto out;

304         if ((typename = builtin_typename(sym))) {
305                 int len = strlen(typename);
306                 if (name->start != name->end)
307                         *--name->start = ' ';
308                 name->start -= len;
309                 memcpy(name->start, typename, len);
310                 goto out;
311         }

313         /* Prepend */
314         switch (sym->type) {
315         case SYM_PTR:
316                 prepend(name, "*");
317                 mod = sym->ctype.modifiers;
318                 as = sym->ctype.as;
319                 was_ptr = 1;
320                 break;

322         case SYM_FN:
323                 if (was_ptr) {
324                         prepend(name, "( ");
```



```
325                         append(name, " )");
326                         was_ptr = 0;
327                 }
328                 append(name, "( ... )");
329                 break;

331         case SYM_STRUCT:
332                 if (name->start != name->end)
333                         *--name->start = ' ';
334                 prepend(name, "struct %s", show_ident(sym->ident));
335                 goto out;

337         case SYM_UNION:
338                 if (name->start != name->end)
339                         *--name->start = ' ';
340                 prepend(name, "union %s", show_ident(sym->ident));
341                 goto out;

343         case SYM_ENUM:
344                 prepend(name, "enum %s ", show_ident(sym->ident));
345                 break;

347         case SYM_NODE:
348                 append(name, "%s", show_ident(sym->ident));
349                 mod |= sym->ctype.modifiers;
350                 as |= sym->ctype.as;
351                 break;

353         case SYM_BITFIELD:
354                 mod |= sym->ctype.modifiers;
355                 as |= sym->ctype.as;
356                 append(name, ":%d", sym->bit_size);
357                 break;

359         case SYM_LABEL:
360                 append(name, "label(%s:%p)", show_ident(sym->ident), sym);
361                 return;

363         case SYM_ARRAY:
364                 mod |= sym->ctype.modifiers;
365                 as |= sym->ctype.as;
366                 if (was_ptr) {
367                         prepend(name, "( ");
368                         append(name, " )");
369                         was_ptr = 0;
370                 }
371                 append(name, "[%lld]", get_expression_value(sym->array_size));
372                 break;

374         case SYM_RESTRICT:
375                 if (!sym->ident) {
376                         restr = 1;
377                         break;
378                 }
379                 if (name->start != name->end)
380                         *--name->start = ' ';
381                 prepend(name, "restricted %s", show_ident(sym->ident));
382                 goto out;

384         case SYM_FOULED:
385                 fouled = 1;
386                 break;

388         default:
389                 if (name->start != name->end)
390                         *--name->start = ' ';
```

```
391                 prepend(name, "unknown type %d", sym->type);
392                 goto out;
393         }

395         sym = sym->ctype.base_type;
396         goto deeper;

398 out:
399         if (restr)
400                 prepend(name, "restricted ");
401         if (fouled)
402                 prepend(name, "fouled ");
403 }

405 void show_type(struct symbol *sym)
406 {
407         char array[200];
408         struct type_name name;

410         name.start = name.end = array+100;
411         do_show_type(sym, &name);
412         *name.end = 0;
413         printf("%s", name.start);
414 }

416 const char *show_typename(struct symbol *sym)
417 {
418         static char array[200];
419         struct type_name name;

421         name.start = name.end = array+100;
422         do_show_type(sym, &name);
423         *name.end = 0;
424         return name.start;
425 }

427 void show_symbol(struct symbol *sym)
428 {
429         struct symbol *type;

431         if (!sym)
432                 return;

434         if (sym->ctype.alignment)
435                 printf(".align %ld\n", sym->ctype.alignment);

437         show_type(sym);
438         type = sym->ctype.base_type;
439         if (!type) {
440                 printf("\n");
441                 return;
442         }

444         /*
445          * Show actual implementation information
446          */
447         switch (type->type) {
448                 struct symbol *member;

450         case SYM_STRUCT:
451         case SYM_UNION:
452                 printf(" {\n");
453                 FOR_EACH_PTR(type->symbol_list, member) {
454                         show_struct_member(member);
455                 } END_FOR_EACH_PTR(member);
456                 printf("}\n");
```



```
457                 break;

459         case SYM_FN: {
460                 struct statement *stmt = type->stmt;
461                 printf("\n");
462                 if (stmt) {
463                         int val;
464                         val = show_statement(stmt);
465                         if (val)
466                                 printf("\tmov.%d\t\tretval,%d\n", stmt->ret->bit
467                         printf("\tret\n");
468                 }
469                 break;
470         }

472         default:
473                 printf("\n");
474                 break;
475         }

477         if (sym->initializer) {
478                 printf(" = \n");
479                 show_expression(sym->initializer);
480         }
481 }

483 static int show_symbol_init(struct symbol *sym);

485 static int new_pseudo(void)
486 {
487         static int nr = 0;
488         return ++nr;
489 }

491 static int new_label(void)
492 {
493         static int label = 0;
494         return ++label;
495 }

497 static void show_switch_statement(struct statement *stmt)
498 {
499         int val = show_expression(stmt->switch_expression);
500         struct symbol *sym;
501         printf("\tswitch v%d\n", val);

503         /*
504          * Debugging only: Check that the case list is correct
505          * by printing it out.
506          *
507          * This is where a _real_ back-end would go through the
508          * cases to decide whether to use a lookup table or a
509          * series of comparisons etc
510          */
511         printf("# case table:\n");
512         FOR_EACH_PTR(stmt->switch_case->symbol_list, sym) {
513                 struct statement *case_stmt = sym->stmt;
514                 struct expression *expr = case_stmt->case_expression;
515                 struct expression *to = case_stmt->case_to;

517                 if (!expr) {
518                         printf("    default");
519                 } else {
520                         if (expr->type == EXPR_VALUE) {
521                                 printf("    case %lld", expr->value);
522                                 if (to) {
```

```
523                                     if (to->type == EXPR_VALUE) {
524                                             printf(" .. %lld", to->value);
525                                     } else {
526                                             printf(" .. what?");
527                                     }
528                             }
529                     } else
530                             printf("    what?");
531             }
532             printf(": .L%p\n", sym);
533     } END_FOR_EACH_PTR(sym);
534     printf("# end case table\n");

536     show_statement(stmt->switch_statement);

538     if (stmt->switch_break->used)
539             printf(".L%p:\n", stmt->switch_break);
540 }

542 static void show_symbol_decl(struct symbol_list *syms)
543 {
544     struct symbol *sym;
545     FOR_EACH_PTR(syms, sym) {
546             show_symbol_init(sym);
547     } END_FOR_EACH_PTR(sym);
548 }

550 static int show_return_stmt(struct statement *stmt);

552 /*
553  * Print out a statement
554  */
555 int show_statement(struct statement *stmt)
556 {
557     if (!stmt)
558             return 0;
559     switch (stmt->type) {
560     case STMT_DECLARATION:
561             show_symbol_decl(stmt->declaration);
562             return 0;
563     case STMT_RETURN:
564             return show_return_stmt(stmt);
565     case STMT_COMPOUND: {
566             struct statement *s;
567             int last = 0;

569             if (stmt->inline_fn) {
570                     show_statement(stmt->args);
571                     printf("\tbegin_inline \t%s\n", show_ident(stmt->inline_
572             }
573             FOR_EACH_PTR(stmt->stmts, s) {
574                     last = show_statement(s);
575             } END_FOR_EACH_PTR(s);
576             if (stmt->ret) {
577                     int addr, bits;
578                     printf(".L%p:\n", stmt->ret);
579                     addr = show_symbol_expr(stmt->ret);
580                     bits = stmt->ret->bit_size;
581                     last = new_pseudo();
582                     printf("\tld.%d\t\tv%d,[v%d]\n", bits, last, addr);
583             }
584             if (stmt->inline_fn)
585                     printf("\tend_inlined\t%s\n", show_ident(stmt->inline_fn
586             return last;
587     }
```



```
589     case STMT_EXPRESSION:
590             return show_expression(stmt->expression);
591     case STMT_IF: {
592             int val, target;
593             struct expression *cond = stmt->if_conditional;

595 /* This is only valid if nobody can jump into the "dead" statement */
596 #if 0
597             if (cond->type == EXPR_VALUE) {
598                     struct statement *s = stmt->if_true;
599                     if (!cond->value)
600                             s = stmt->if_false;
601                     show_statement(s);
602                     break;
603             }
604 #endif
605             val = show_expression(cond);
606             target = new_label();
607             printf("\tje\t\tv%d,.L%d\n", val, target);
608             show_statement(stmt->if_true);
609             if (stmt->if_false) {
610                     int last = new_label();
611                     printf("\tjmp\t\t.L%d\n", last);
612                     printf(".L%d:\n", target);
613                     target = last;
614                     show_statement(stmt->if_false);
615             }
616             printf(".L%d:\n", target);
617             break;
618     }
619     case STMT_SWITCH:
620             show_switch_statement(stmt);
621             break;

623     case STMT_CASE:
624             printf(".L%p:\n", stmt->case_label);
625             show_statement(stmt->case_statement);
626             break;

628     case STMT_ITERATOR: {
629             struct statement  *pre_statement = stmt->iterator_pre_statement;
630             struct expression *pre_condition = stmt->iterator_pre_condition;
631             struct statement  *statement = stmt->iterator_statement;
632             struct statement  *post_statement = stmt->iterator_post_statemen
633             struct expression *post_condition = stmt->iterator_post_conditio
634             int val, loop_top = 0, loop_bottom = 0;

636             show_symbol_decl(stmt->iterator_syms);
637             show_statement(pre_statement);
638             if (pre_condition) {
639                     if (pre_condition->type == EXPR_VALUE) {
640                             if (!pre_condition->value) {
641                                     loop_bottom = new_label();
642                                     printf("\tjmp\t\t.L%d\n", loop_bottom);
643                             }
644                     } else {
645                             loop_bottom = new_label();
646                             val = show_expression(pre_condition);
647                             printf("\tje\t\tv%d, .L%d\n", val, loop_bottom);
648                     }
649             }
650             if (!post_condition || post_condition->type != EXPR_VALUE || pos
651                     loop_top = new_label();
652                     printf(".L%d:\n", loop_top);
653             }
654             show_statement(statement);
```

```
655                 if (stmt->iterator_continue->used)
656                         printf(".L%p:\n", stmt->iterator_continue);
657                 show_statement(post_statement);
658                 if (!post_condition) {
659                         printf("\tjmp\t\t.L%d\n", loop_top);
660                 } else if (post_condition->type == EXPR_VALUE) {
661                         if (post_condition->value)
662                                 printf("\tjmp\t\t.L%d\n", loop_top);
663                 } else {
664                         val = show_expression(post_condition);
665                         printf("\tjne\t\tv%d, .L%d\n", val, loop_top);
666                 }
667                 if (stmt->iterator_break->used)
668                         printf(".L%p:\n", stmt->iterator_break);
669                 if (loop_bottom)
670                         printf(".L%d:\n", loop_bottom);
671                 break;
672         }
673         case STMT_NONE:
674                 break;
675
676         case STMT_LABEL:
677                 printf(".L%p:\n", stmt->label_identifier);
678                 show_statement(stmt->label_statement);
679                 break;
680
681         case STMT_GOTO:
682                 if (stmt->goto_expression) {
683                         int val = show_expression(stmt->goto_expression);
684                         printf("\tgoto\t\t*v%d\n", val);
685                 } else {
686                         printf("\tgoto\t\t.L%p\n", stmt->goto_label);
687                 }
688                 break;
689         case STMT_ASM:
690                 printf("\tasm( .... )\n");
691                 break;
692         case STMT_CONTEXT: {
693                 int val = show_expression(stmt->expression);
694                 printf("\tcontext( %d )\n", val);
695                 break;
696         }
697         case STMT_RANGE: {
698                 int val = show_expression(stmt->range_expression);
699                 int low = show_expression(stmt->range_low);
700                 int high = show_expression(stmt->range_high);
701                 printf("\trange( %d %d-%d)\n", val, low, high);
702                 break;
703         }
704         }
705         return 0;
706 }
707
708 static int show_call_expression(struct expression *expr)
709 {
710         struct symbol *direct;
711         struct expression *arg, *fn;
712         int fncall, retval;
713         int framesize;
714
715         if (!expr->ctype) {
716                 warning(expr->pos, "\tcall with no type!");
717                 return 0;
718         }
719
720         framesize = 0;
```



```
721         FOR_EACH_PTR_REVERSE(expr->args, arg) {
722                 int new = show_expression(arg);
723                 int size = arg->ctype->bit_size;
724                 printf("\tpush.%d\t\tv%d\n", size, new);
725                 framesize += bits_to_bytes(size);
726         } END_FOR_EACH_PTR_REVERSE(arg);
727
728         fn = expr->fn;
729
730         /* Remove dereference, if any */
731         direct = NULL;
732         if (fn->type == EXPR_PREOP) {
733                 if (fn->unop->type == EXPR_SYMBOL) {
734                         struct symbol *sym = fn->unop->symbol;
735                         if (sym->ctype.base_type->type == SYM_FN)
736                                 direct = sym;
737                 }
738         }
739         if (direct) {
740                 printf("\tcall\t\t%s\n", show_ident(direct->ident));
741         } else {
742                 fncall = show_expression(fn);
743                 printf("\tcall\t\t*v%d\n", fncall);
744         }
745         if (framesize)
746                 printf("\tadd.%d\t\tvSP,vSP,$%d\n", bits_in_pointer, framesize);
747
748         retval = new_pseudo();
749         printf("\tmov.%d\t\tv%d,retval\n", expr->ctype->bit_size, retval);
750         return retval;
751 }
752
753 static int show_comma(struct expression *expr)
754 {
755         show_expression(expr->left);
756         return show_expression(expr->right);
757 }
758
759 static int show_binop(struct expression *expr)
760 {
761         int left = show_expression(expr->left);
762         int right = show_expression(expr->right);
763         int new = new_pseudo();
764         const char *opname;
765         static const char *name[] = {
766                 ['+'] = "add", ['-'] = "sub",
767                 ['*'] = "mul", ['/'] = "div",
768                 ['%'] = "mod", ['&'] = "and",
769                 ['|'] = "lor", ['^'] = "xor"
770         };
771         unsigned int op = expr->op;
772
773         opname = show_special(op);
774         if (op < ARRAY_SIZE(name))
775                 opname = name[op];
776         printf("\t%s.%d\t\tv%d,v%d,v%d\n", opname,
777                 expr->ctype->bit_size,
778                 new, left, right);
779         return new;
780 }
781
782 static int show_slice(struct expression *expr)
783 {
784         int target = show_expression(expr->base);
785         int new = new_pseudo();
786         printf("\tslice.%d\t\tv%d,v%d,%d\n", expr->r_nrbits, target, new, expr->
```

```
787         return new;
788 }

790 static int show_regular_preop(struct expression *expr)
791 {
792         int target = show_expression(expr->unop);
793         int new = new_pseudo();
794         static const char *name[] = {
795                 ['!'] = "nonzero", ['-'] = "neg",
796                 ['~'] = "not",
797         };
798         unsigned int op = expr->op;
799         const char *opname;

801         opname = show_special(op);
802         if (op < ARRAY_SIZE(name))
803                 opname = name[op];
804         printf("\t%s.%d\t\tv%d,v%d\n", opname, expr->ctype->bit_size, new, targe
805         return new;
806 }

808 /*
809  * FIXME! Not all accesses are memory loads. We should
810  * check what kind of symbol is behind the dereference.
811  */
812 static int show_address_gen(struct expression *expr)
813 {
814         return show_expression(expr->unop);
815 }

817 static int show_load_gen(int bits, struct expression *expr, int addr)
818 {
819         int new = new_pseudo();

821         printf("\tld.%d\t\tv%d,[v%d]\n", bits, new, addr);
822         return new;
823 }

825 static void show_store_gen(int bits, int value, struct expression *expr, int add
826 {
827         /* FIXME!!! Bitfield store! */
828         printf("\tst.%d\t\tv%d,[v%d]\n", bits, value, addr);
829 }

831 static int show_assignment(struct expression *expr)
832 {
833         struct expression *target = expr->left;
834         int val, addr, bits;

836         if (!expr->ctype)
837                 return 0;

839         bits = expr->ctype->bit_size;
840         val = show_expression(expr->right);
841         addr = show_address_gen(target);
842         show_store_gen(bits, val, target, addr);
843         return val;
844 }

846 static int show_return_stmt(struct statement *stmt)
847 {
848         struct expression *expr = stmt->ret_value;
849         struct symbol *target = stmt->ret_target;

851         if (expr && expr->ctype) {
852                 int val = show_expression(expr);
```



```
853                 int bits = expr->ctype->bit_size;
854                 int addr = show_symbol_expr(target);
855                 show_store_gen(bits, val, NULL, addr);
856         }
857         printf("\tret\t\t(%p)\n", target);
858         return 0;
859 }

861 static int show_initialization(struct symbol *sym, struct expression *expr)
862 {
863         int val, addr, bits;

865         if (!expr->ctype)
866                 return 0;

868         bits = expr->ctype->bit_size;
869         val = show_expression(expr);
870         addr = show_symbol_expr(sym);
871         // FIXME! The "target" expression is for bitfield store information.
872         // Leave it NULL, which works fine.
873         show_store_gen(bits, val, NULL, addr);
874         return 0;
875 }

877 static int show_access(struct expression *expr)
878 {
879         int addr = show_address_gen(expr);
880         return show_load_gen(expr->ctype->bit_size, expr, addr);
881 }

883 static int show_inc_dec(struct expression *expr, int postop)
884 {
885         int addr = show_address_gen(expr->unop);
886         int retval, new;
887         const char *opname = expr->op == SPECIAL_INCREMENT ? "add" : "sub";
888         int bits = expr->ctype->bit_size;

890         retval = show_load_gen(bits, expr->unop, addr);
891         new = retval;
892         if (postop)
893                 new = new_pseudo();
894         printf("\t%s.%d\t\tv%d,v%d,$1\n", opname, bits, new, retval);
895         show_store_gen(bits, new, expr->unop, addr);
896         return retval;
897 }

899 static int show_preop(struct expression *expr)
900 {
901         /*
902          * '*' is an lvalue access, and is fundamentally different
903          * from an arithmetic operation. Maybe it should have an
904          * expression type of its own..
905          */
906         if (expr->op == '*')
907                 return show_access(expr);
908         if (expr->op == SPECIAL_INCREMENT || expr->op == SPECIAL_DECREMENT)
909                 return show_inc_dec(expr, 0);
910         return show_regular_preop(expr);
911 }

913 static int show_postop(struct expression *expr)
914 {
915         return show_inc_dec(expr, 1);
916 }

918 static int show_symbol_expr(struct symbol *sym)
```

```
{
        int new = new_pseudo();

        if (sym->initializer && sym->initializer->type == EXPR_STRING)
                return show_string_expr(sym->initializer);

        if (sym->ctype.modifiers & (MOD_TOPLEVEL | MOD_EXTERN | MOD_STATIC)) {
                printf("\tmovi.%d\t\tv%d,$%s\n", bits_in_pointer, new, show_iden
                return new;
        }
        if (sym->ctype.modifiers & MOD_ADDRESSABLE) {
                printf("\taddi.%d\t\tv%d,vFP,$%lld\n", bits_in_pointer, new, sym
                return new;
        }
        printf("\taddi.%d\t\tv%d,vFP,$offsetof(%s:%p)\n", bits_in_pointer, new,
        return new;
}

static int show_symbol_init(struct symbol *sym)
{
        struct expression *expr = sym->initializer;

        if (expr) {
                int val, addr, bits;

                bits = expr->ctype->bit_size;
                val = show_expression(expr);
                addr = show_symbol_expr(sym);
                show_store_gen(bits, val, NULL, addr);
        }
        return 0;
}

static int type_is_signed(struct symbol *sym)
{
        if (sym->type == SYM_NODE)
                sym = sym->ctype.base_type;
        if (sym->type == SYM_PTR)
                return 0;
        return !(sym->ctype.modifiers & MOD_UNSIGNED);
}

static int show_cast_expr(struct expression *expr)
{
        struct symbol *old_type, *new_type;
        int op = show_expression(expr->cast_expression);
        int oldbits, newbits;
        int new, is_signed;

        old_type = expr->cast_expression->ctype;
        new_type = expr->cast_type;

        oldbits = old_type->bit_size;
        newbits = new_type->bit_size;
        if (oldbits >= newbits)
                return op;
        new = new_pseudo();
        is_signed = type_is_signed(old_type);
        if (is_signed) {
                printf("\tsext%d.%d\tv%d,v%d\n", oldbits, newbits, new, op);
        } else {
                printf("\tandl.%d\t\tv%d,v%d,$%lu\n", newbits, new, op, (1UL <<
        }
        return new;
}
```



```
static int show_value(struct expression *expr)
{
        int new = new_pseudo();
        unsigned long long value = expr->value;

        printf("\tmovi.%d\t\tv%d,$%llu\n", expr->ctype->bit_size, new, value);
        return new;
}

static int show_fvalue(struct expression *expr)
{
        int new = new_pseudo();
        long double value = expr->fvalue;

        printf("\tmovf.%d\t\tv%d,$%Lf\n", expr->ctype->bit_size, new, value);
        return new;
}

static int show_string_expr(struct expression *expr)
{
        int new = new_pseudo();

        printf("\tmovi.%d\t\tv%d,&%s\n", bits_in_pointer, new, show_string(expr-
        return new;
}

static int show_label_expr(struct expression *expr)
{
        int new = new_pseudo();
        printf("\tmovi.%d\t\tv%d,.L%p\n",bits_in_pointer, new, expr->label_symbo
        return new;
}

static int show_conditional_expr(struct expression *expr)
{
        int cond = show_expression(expr->conditional);
        int true = show_expression(expr->cond_true);
        int false = show_expression(expr->cond_false);
        int new = new_pseudo();

        printf("[v%d]\tcmov.%d\t\tv%d,v%d,v%d\n", cond, expr->ctype->bit_size, n
        return new;
}

static int show_statement_expr(struct expression *expr)
{
        return show_statement(expr->statement);
}

static int show_position_expr(struct expression *expr, struct symbol *base)
{
        int new = show_expression(expr->init_expr);
        struct symbol *ctype = expr->init_expr->ctype;
        int bit_offset;

        bit_offset = ctype ? ctype->bit_offset : -1;

        printf("\tinsert v%d at [%d:%d] of %s\n", new,
                expr->init_offset, bit_offset,
                show_ident(base->ident));
        return 0;
}

static int show_initializer_expr(struct expression *expr, struct symbol *ctype)
{
        struct expression *entry;
```

```
1052         FOR_EACH_PTR(expr->expr_list, entry) {

1054 again:
1055                 // Nested initializers have their positions already
1056                 // recursively calculated - just output them too
1057                 if (entry->type == EXPR_INITIALIZER) {
1058                         show_initializer_expr(entry, ctype);
1059                         continue;
1060                 }

1062                 // Initializer indexes and identifiers should
1063                 // have been evaluated to EXPR_POS
1064                 if (entry->type == EXPR_IDENTIFIER) {
1065                         printf(" AT '%s':\n", show_ident(entry->expr_ident));
1066                         entry = entry->ident_expression;
1067                         goto again;
1068                 }
1069
1070                 if (entry->type == EXPR_INDEX) {
1071                         printf(" AT '%d..%d:\n", entry->idx_from, entry->idx_to)
1072                         entry = entry->idx_expression;
1073                         goto again;
1074                 }
1075                 if (entry->type == EXPR_POS) {
1076                         show_position_expr(entry, ctype);
1077                         continue;
1078                 }
1079                 show_initialization(ctype, entry);
1080         } END_FOR_EACH_PTR(entry);
1081         return 0;
1082 }

1084 int show_symbol_expr_init(struct symbol *sym)
1085 {
1086         struct expression *expr = sym->initializer;

1088         if (expr)
1089                 show_expression(expr);
1090         return show_symbol_expr(sym);
1091 }

1093 /*
1094  * Print out an expression. Return the pseudo that contains the
1095  * variable.
1096  */
1097 int show_expression(struct expression *expr)
1098 {
1099         if (!expr)
1100                 return 0;

1102         if (!expr->ctype) {
1103                 struct position *pos = &expr->pos;
1104                 printf("\tno type at %s:%d:%d\n",
1105                         stream_name(pos->stream),
1106                         pos->line, pos->pos);
1107                 return 0;
1108         }
1109
1110         switch (expr->type) {
1111         case EXPR_CALL:
1112                 return show_call_expression(expr);
1113
1114         case EXPR_ASSIGNMENT:
1115                 return show_assignment(expr);
```



```
1117         case EXPR_COMMA:
1118                 return show_comma(expr);
1119         case EXPR_BINOP:
1120         case EXPR_COMPARE:
1121         case EXPR_LOGICAL:
1122                 return show_binop(expr);
1123         case EXPR_PREOP:
1124                 return show_preop(expr);
1125         case EXPR_POSTOP:
1126                 return show_postop(expr);
1127         case EXPR_SYMBOL:
1128                 return show_symbol_expr(expr->symbol);
1129         case EXPR_DEREF:
1130         case EXPR_SIZEOF:
1131         case EXPR_PTRSIZEOF:
1132         case EXPR_ALIGNOF:
1133         case EXPR_OFFSETOF:
1134                 warning(expr->pos, "invalid expression after evaluation");
1135                 return 0;
1136         case EXPR_CAST:
1137         case EXPR_FORCE_CAST:
1138         case EXPR_IMPLIED_CAST:
1139                 return show_cast_expr(expr);
1140         case EXPR_VALUE:
1141                 return show_value(expr);
1142         case EXPR_FVALUE:
1143                 return show_fvalue(expr);
1144         case EXPR_STRING:
1145                 return show_string_expr(expr);
1146         case EXPR_INITIALIZER:
1147                 return show_initializer_expr(expr, expr->ctype);
1148         case EXPR_SELECT:
1149         case EXPR_CONDITIONAL:
1150                 return show_conditional_expr(expr);
1151         case EXPR_STATEMENT:
1152                 return show_statement_expr(expr);
1153         case EXPR_LABEL:
1154                 return show_label_expr(expr);
1155         case EXPR_SLICE:
1156                 return show_slice(expr);

1158         // None of these should exist as direct expressions: they are only
1159         // valid as sub-expressions of initializers.
1160         case EXPR_POS:
1161                 warning(expr->pos, "unable to show plain initializer position ex
1162                 return 0;
1163         case EXPR_IDENTIFIER:
1164                 warning(expr->pos, "unable to show identifier expression");
1165                 return 0;
1166         case EXPR_INDEX:
1167                 warning(expr->pos, "unable to show index expression");
1168                 return 0;
1169         case EXPR_TYPE:
1170                 warning(expr->pos, "unable to show type expression");
1171                 return 0;
1172         }
1173         return 0;
1174 }
```

```
**********************************************************
   26644 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/simplify.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 /*
 2  * Simplify - do instruction simplification before CSE
 3  *
 4  * Copyright (C) 2004 Linus Torvalds
 5  */

 7 #include <assert.h>

 9 #include "parse.h"
10 #include "expression.h"
11 #include "linearize.h"
12 #include "flow.h"
13 #include "symbol.h"

15 /* Find the trivial parent for a phi-source */
16 static struct basic_block *phi_parent(struct basic_block *source, pseudo_t pseud
17 {
18         /* Can't go upwards if the pseudo is defined in the bb it came from.. */
19         if (pseudo->type == PSEUDO_REG) {
20                 struct instruction *def = pseudo->def;
21                 if (def->bb == source)
22                         return source;
23         }
24         if (bb_list_size(source->children) != 1 || bb_list_size(source->parents)
25                 return source;
26         return first_basic_block(source->parents);
27 }

29 /*
30  * Copy the phi-node's phisrcs into to given array.
31  * Returns 0 if the the list contained the expected
32  * number of element, a positive number if there was
33  * more than expected and a negative one if less.
34  *
35  * Note: we can't reuse a function like linearize_ptr_list()
36  * because any VOIDs in the phi-list must be ignored here
37  * as in this context they mean 'entry has been removed'.
38  */
39 static int get_phisources(struct instruction *sources[], int nbr, struct instruc
40 {
41         pseudo_t phi;
42         int i = 0;

44         assert(insn->opcode == OP_PHI);
45         FOR_EACH_PTR(insn->phi_list, phi) {
46                 struct instruction *def;
47                 if (phi == VOID)
48                         continue;
49                 if (i >= nbr)
50                         return 1;
51                 def = phi->def;
52                 assert(def->opcode == OP_PHISOURCE);
53                 sources[i++] = def;
54         } END_FOR_EACH_PTR(phi);
55         return i - nbr;
56 }

58 static int if_convert_phi(struct instruction *insn)
59 {
60         struct instruction *array[2];
```



```
61         struct basic_block *parents[3];
62         struct basic_block *bb, *bb1, *bb2, *source;
63         struct instruction *br;
64         pseudo_t p1, p2;

66         bb = insn->bb;
67         if (get_phisources(array, 2, insn))
68                 return 0;
69         if (linearize_ptr_list((struct ptr_list *)bb->parents, (void **)parents,
70                 return 0;
71         p1 = array[0]->src1;
72         bb1 = array[0]->bb;
73         p2 = array[1]->src1;
74         bb2 = array[1]->bb;

76         /* Only try the simple "direct parents" case */
77         if ((bb1 != parents[0] || bb2 != parents[1]) &&
78             (bb1 != parents[1] || bb2 != parents[0]))
79                 return 0;

81         /*
82          * See if we can find a common source for this..
83          */
84         source = phi_parent(bb1, p1);
85         if (source != phi_parent(bb2, p2))
86                 return 0;

88         /*
89          * Cool. We now know that 'source' is the exclusive
90          * parent of both phi-nodes, so the exit at the
91          * end of it fully determines which one it is, and
92          * we can turn it into a select.
93          *
94          * HOWEVER, right now we only handle regular
95          * conditional branches. No multijumps or computed
96          * stuff. Verify that here.
97          */
98         br = last_instruction(source->insns);
99         if (!br || br->opcode != OP_CBR)
100                return 0;

102        assert(br->cond);
103        assert(br->bb_false);

105        /*
106         * We're in business. Match up true/false with p1/p2.
107         */
108        if (br->bb_true == bb2 || br->bb_false == bb1) {
109                pseudo_t p = p1;
110                p1 = p2;
111                p2 = p;
112        }

114        /*
115         * OK, we can now replace that last
116         *
117         *      br cond, a, b
118         *
119         * with the sequence
120         *
121         *      setcc cond
122         *      select pseudo, p1, p2
123         *      br cond, a, b
124         *
125         * and remove the phi-node. If it then
126         * turns out that 'a' or 'b' is entirely
```

```
127          * empty (common case), and now no longer
128          * a phi-source, we'll be able to simplify
129          * the conditional branch too.
130          */
131         insert_select(source, br, insn, p1, p2);
132         kill_instruction(insn);
133         return REPEAT_CSE;
134 }

136 static int clean_up_phi(struct instruction *insn)
137 {
138         pseudo_t phi;
139         struct instruction *last;
140         int same;

142         last = NULL;
143         same = 1;
144         FOR_EACH_PTR(insn->phi_list, phi) {
145                 struct instruction *def;
146                 if (phi == VOID)
147                         continue;
148                 def = phi->def;
149                 if (def->src1 == VOID || !def->bb)
150                         continue;
151                 if (last) {
152                         if (last->src1 != def->src1)
153                                 same = 0;
154                         continue;
155                 }
156                 last = def;
157         } END_FOR_EACH_PTR(phi);

159         if (same) {
160                 pseudo_t pseudo = last ? last->src1 : VOID;
161                 convert_instruction_target(insn, pseudo);
162                 kill_instruction(insn);
163                 return REPEAT_CSE;
164         }

166         return if_convert_phi(insn);
167 }

169 static int delete_pseudo_user_list_entry(struct pseudo_user_list **list, pseudo_
170 {
171         struct pseudo_user *pu;

173         FOR_EACH_PTR(*list, pu) {
174                 if (pu->userp == entry) {
175                         MARK_CURRENT_DELETED(pu);
176                         if (!--count)
177                                 goto out;
178                 }
179         } END_FOR_EACH_PTR(pu);
180         assert(count <= 0);
181 out:
182         if (ptr_list_size((struct ptr_list *) *list) == 0)
183                 *list = NULL;
184         return count;
185 }

187 static inline void remove_usage(pseudo_t p, pseudo_t *usep)
188 {
189         if (has_use_list(p)) {
190                 delete_pseudo_user_list_entry(&p->users, usep, 1);
191                 if (!p->users)
192                         kill_instruction(p->def);
```



```
193         }
194 }

196 void kill_use(pseudo_t *usep)
197 {
198         if (usep) {
199                 pseudo_t p = *usep;
200                 *usep = VOID;
201                 remove_usage(p, usep);
202         }
203 }

205 static void kill_use_list(struct pseudo_list *list)
206 {
207         pseudo_t p;
208         FOR_EACH_PTR(list, p) {
209                 if (p == VOID)
210                         continue;
211                 kill_use(THIS_ADDRESS(p));
212         } END_FOR_EACH_PTR(p);
213 }

215 /*
216  * kill an instruction:
217  * - remove it from its bb
218  * - remove the usage of all its operands
219  * If forse is zero, the normal case, the function only for
220  * instructions free of (possible) side-effects. Otherwise
221  * the function does that unconditionally (must only be used
222  * for unreachable instructions.
223  */
224 void kill_insn(struct instruction *insn, int force)
225 {
226         if (!insn || !insn->bb)
227                 return;

229         switch (insn->opcode) {
230         case OP_SEL:
231         case OP_RANGE:
232                 kill_use(&insn->src3);
233                 /* fall through */

235         case OP_BINARY ... OP_BINCMP_END:
236                 kill_use(&insn->src2);
237                 /* fall through */

239         case OP_CAST:
240         case OP_SCAST:
241         case OP_FPCAST:
242         case OP_PTRCAST:
243         case OP_SETVAL:
244         case OP_NOT: case OP_NEG:
245         case OP_SLICE:
246                 kill_use(&insn->src1);
247                 break;

249         case OP_PHI:
250                 kill_use_list(insn->phi_list);
251                 break;
252         case OP_PHISOURCE:
253                 kill_use(&insn->phi_src);
254                 break;

256         case OP_SYMADDR:
257                 repeat_phase |= REPEAT_SYMBOL_CLEANUP;
258                 break;
```

```
260         case OP_CBR:
261         case OP_COMPUTEDGOTO:
262                 kill_use(&insn->cond);
263                 break;
264
265         case OP_CALL:
266                 if (!force) {
267                         /* a "pure" function can be killed too */
268                         if (!(insn->func->type == PSEUDO_SYM))
269                                 return;
270                         if (!(insn->func->sym->ctype.modifiers & MOD_PURE))
271                                 return;
272                 }
273                 kill_use_list(insn->arguments);
274                 if (insn->func->type == PSEUDO_REG)
275                         kill_use(&insn->func);
276                 break;
277
278         case OP_LOAD:
279                 if (!force && insn->type->ctype.modifiers & MOD_VOLATILE)
280                         return;
281                 kill_use(&insn->src);
282                 break;
283
284         case OP_STORE:
285                 if (!force)
286                         return;
287                 kill_use(&insn->src);
288                 kill_use(&insn->target);
289                 break;
290
291         case OP_ENTRY:
292                 /* ignore */
293                 return;
294
295         case OP_BR:
296         default:
297                 break;
298         }
299
300         insn->bb = NULL;
301         repeat_phase |= REPEAT_CSE;
302         return;
303 }
304
305 /*
306  * Kill trivially dead instructions
307  */
308 static int dead_insn(struct instruction *insn, pseudo_t *src1, pseudo_t *src2, p
309 {
310         struct pseudo_user *pu;
311         FOR_EACH_PTR(insn->target->users, pu) {
312                 if (*pu->userp != VOID)
313                         return 0;
314         } END_FOR_EACH_PTR(pu);
315
316         insn->bb = NULL;
317         kill_use(src1);
318         kill_use(src2);
319         kill_use(src3);
320         return REPEAT_CSE;
321 }
322
323 static inline int constant(pseudo_t pseudo)
324 {
```



```
325         return pseudo->type == PSEUDO_VAL;
326 }
327
328 static int replace_with_pseudo(struct instruction *insn, pseudo_t pseudo)
329 {
330         convert_instruction_target(insn, pseudo);
331
332         switch (insn->opcode) {
333         case OP_SEL:
334         case OP_RANGE:
335                 kill_use(&insn->src3);
336         case OP_BINARY ... OP_BINCMP_END:
337                 kill_use(&insn->src2);
338         case OP_NOT:
339         case OP_NEG:
340         case OP_SYMADDR:
341         case OP_CAST:
342         case OP_SCAST:
343         case OP_FPCAST:
344         case OP_PTRCAST:
345                 kill_use(&insn->src1);
346                 break;
347
348         default:
349                 assert(0);
350         }
351         insn->bb = NULL;
352         return REPEAT_CSE;
353 }
354
355 unsigned int value_size(long long value)
356 {
357         value >>= 8;
358         if (!value)
359                 return 8;
360         value >>= 8;
361         if (!value)
362                 return 16;
363         value >>= 16;
364         if (!value)
365                 return 32;
366         return 64;
367 }
368
369 /*
370  * Try to determine the maximum size of bits in a pseudo.
371  *
372  * Right now this only follow casts and constant values, but we
373  * could look at things like logical 'and' instructions etc.
374  */
375 static unsigned int operand_size(struct instruction *insn, pseudo_t pseudo)
376 {
377         unsigned int size = insn->size;
378
379         if (pseudo->type == PSEUDO_REG) {
380                 struct instruction *src = pseudo->def;
381                 if (src && src->opcode == OP_CAST && src->orig_type) {
382                         unsigned int orig_size = src->orig_type->bit_size;
383                         if (orig_size < size)
384                                 size = orig_size;
385                 }
386         }
387         if (pseudo->type == PSEUDO_VAL) {
388                 unsigned int orig_size = value_size(pseudo->value);
389                 if (orig_size < size)
390                         size = orig_size;
```

```
391         }
392         return size;
393 }

395 static int simplify_asr(struct instruction *insn, pseudo_t pseudo, long long val
396 {
397         unsigned int size = operand_size(insn, pseudo);

399         if (value >= size) {
400                 warning(insn->pos, "right shift by bigger than source value");
401                 return replace_with_pseudo(insn, value_pseudo(insn->type, 0));
402         }
403         if (!value)
404                 return replace_with_pseudo(insn, pseudo);
405         return 0;
406 }

408 static int simplify_mul_div(struct instruction *insn, long long value)
409 {
410         unsigned long long sbit = 1ULL << (insn->size - 1);
411         unsigned long long bits = sbit | (sbit - 1);

413         if (value == 1)
414                 return replace_with_pseudo(insn, insn->src1);

416         switch (insn->opcode) {
417         case OP_MULS:
418         case OP_MULU:
419                 if (value == 0)
420                         return replace_with_pseudo(insn, insn->src2);
421         /* Fall through */
422         case OP_DIVS:
423                 if (!(value & sbit))    // positive
424                         break;

426                 value |= ~bits;
427                 if (value == -1) {
428                         insn->opcode = OP_NEG;
429                         return REPEAT_CSE;
430                 }
431         }

433         return 0;
434 }

436 static int compare_opcode(int opcode, int inverse)
437 {
438         if (!inverse)
439                 return opcode;

441         switch (opcode) {
442         case OP_SET_EQ: return OP_SET_NE;
443         case OP_SET_NE: return OP_SET_EQ;

445         case OP_SET_LT: return OP_SET_GE;
446         case OP_SET_LE: return OP_SET_GT;
447         case OP_SET_GT: return OP_SET_LE;
448         case OP_SET_GE: return OP_SET_LT;

450         case OP_SET_A:  return OP_SET_BE;
451         case OP_SET_AE: return OP_SET_B;
452         case OP_SET_B:  return OP_SET_AE;
453         case OP_SET_BE: return OP_SET_A;

455         default:
456                 return opcode;
```



```
457         }
458 }

460 static int simplify_seteq_setne(struct instruction *insn, long long value)
461 {
462         pseudo_t old = insn->src1;
463         struct instruction *def = old->def;
464         pseudo_t src1, src2;
465         int inverse;
466         int opcode;

468         if (value != 0 && value != 1)
469                 return 0;

471         if (!def)
472                 return 0;

474         inverse = (insn->opcode == OP_SET_NE) == value;
475         opcode = def->opcode;
476         switch (opcode) {
477         case OP_BINCMP ... OP_BINCMP_END:
478                 // Convert:
479                 //      setcc.n %t <- %a, %b
480                 //      setne.m %r <- %t, $0
481                 // into:
482                 //      setcc.n %t <- %a, %b
483                 //      setcc.m %r <- %a, $b
484                 // and similar for setne/eq ... 0/1
485                 src1 = def->src1;
486                 src2 = def->src2;
487                 insn->opcode = compare_opcode(opcode, inverse);
488                 use_pseudo(insn, src1, &insn->src1);
489                 use_pseudo(insn, src2, &insn->src2);
490                 remove_usage(old, &insn->src1);
491                 return REPEAT_CSE;

493         default:
494                 return 0;
495         }
496 }

498 static int simplify_constant_rightside(struct instruction *insn)
499 {
500         long long value = insn->src2->value;

502         switch (insn->opcode) {
503         case OP_OR_BOOL:
504                 if (value == 1)
505                         return replace_with_pseudo(insn, insn->src2);
506                 goto case_neutral_zero;

508         case OP_SUB:
509                 if (value) {
510                         insn->opcode = OP_ADD;
511                         insn->src2 = value_pseudo(insn->type, -value);
512                         return REPEAT_CSE;
513                 }
514         /* Fall through */
515         case OP_ADD:
516         case OP_OR: case OP_XOR:
517         case OP_SHL:
518         case OP_LSR:
519         case_neutral_zero:
520                 if (!value)
521                         return replace_with_pseudo(insn, insn->src1);
522                 return 0;
```

```
523         case OP_ASR:
524                 return simplify_asr(insn, insn->src1, value);

526         case OP_MODU: case OP_MODS:
527                 if (value == 1)
528                         return replace_with_pseudo(insn, value_pseudo(insn->type
529                 return 0;

531         case OP_DIVU: case OP_DIVS:
532         case OP_MULU: case OP_MULS:
533                 return simplify_mul_div(insn, value);

535         case OP_AND_BOOL:
536                 if (value == 1)
537                         return replace_with_pseudo(insn, insn->src1);
538         /* Fall through */
539         case OP_AND:
540                 if (!value)
541                         return replace_with_pseudo(insn, insn->src2);
542                 return 0;

544         case OP_SET_NE:
545         case OP_SET_EQ:
546                 return simplify_seteq_setne(insn, value);
547         }
548         return 0;
549 }

551 static int simplify_constant_leftside(struct instruction *insn)
552 {
553         long long value = insn->src1->value;

555         switch (insn->opcode) {
556         case OP_ADD: case OP_OR: case OP_XOR:
557                 if (!value)
558                         return replace_with_pseudo(insn, insn->src2);
559                 return 0;

561         case OP_SHL:
562         case OP_LSR: case OP_ASR:
563         case OP_AND:
564         case OP_MULU: case OP_MULS:
565                 if (!value)
566                         return replace_with_pseudo(insn, insn->src1);
567                 return 0;
568         }
569         return 0;
570 }

572 static int simplify_constant_binop(struct instruction *insn)
573 {
574         /* FIXME! Verify signs and sizes!! */
575         long long left = insn->src1->value;
576         long long right = insn->src2->value;
577         unsigned long long ul, ur;
578         long long res, mask, bits;

580         mask = 1ULL << (insn->size-1);
581         bits = mask | (mask-1);

583         if (left & mask)
584                 left |= ~bits;
585         if (right & mask)
586                 right |= ~bits;
587         ul = left & bits;
588         ur = right & bits;
```



```
590         switch (insn->opcode) {
591         case OP_ADD:
592                 res = left + right;
593                 break;
594         case OP_SUB:
595                 res = left - right;
596                 break;
597         case OP_MULU:
598                 res = ul * ur;
599                 break;
600         case OP_MULS:
601                 res = left * right;
602                 break;
603         case OP_DIVU:
604                 if (!ur)
605                         return 0;
606                 res = ul / ur;
607                 break;
608         case OP_DIVS:
609                 if (!right)
610                         return 0;
611                 if (left == mask && right == -1)
612                         return 0;
613                 res = left / right;
614                 break;
615         case OP_MODU:
616                 if (!ur)
617                         return 0;
618                 res = ul % ur;
619                 break;
620         case OP_MODS:
621                 if (!right)
622                         return 0;
623                 if (left == mask && right == -1)
624                         return 0;
625                 res = left % right;
626                 break;
627         case OP_SHL:
628                 res = left << right;
629                 break;
630         case OP_LSR:
631                 res = ul >> ur;
632                 break;
633         case OP_ASR:
634                 res = left >> right;
635                 break;
636        /* Logical */
637         case OP_AND:
638                 res = left & right;
639                 break;
640         case OP_OR:
641                 res = left | right;
642                 break;
643         case OP_XOR:
644                 res = left ^ right;
645                 break;
646         case OP_AND_BOOL:
647                 res = left && right;
648                 break;
649         case OP_OR_BOOL:
650                 res = left || right;
651                 break;
652
653         /* Binary comparison */
654         case OP_SET_EQ:
```

```
655                 res = left == right;
656                 break;
657         case OP_SET_NE:
658                 res = left != right;
659                 break;
660         case OP_SET_LE:
661                 res = left <= right;
662                 break;
663         case OP_SET_GE:
664                 res = left >= right;
665                 break;
666         case OP_SET_LT:
667                 res = left < right;
668                 break;
669         case OP_SET_GT:
670                 res = left > right;
671                 break;
672         case OP_SET_B:
673                 res = ul < ur;
674                 break;
675         case OP_SET_A:
676                 res = ul > ur;
677                 break;
678         case OP_SET_BE:
679                 res = ul <= ur;
680                 break;
681         case OP_SET_AE:
682                 res = ul >= ur;
683                 break;
684         default:
685                 return 0;
686         }
687         res &= bits;

689         replace_with_pseudo(insn, value_pseudo(insn->type, res));
690         return REPEAT_CSE;
691 }

693 static int simplify_binop_same_args(struct instruction *insn, pseudo_t arg)
694 {
695         switch (insn->opcode) {
696         case OP_SET_NE:
697         case OP_SET_LT: case OP_SET_GT:
698         case OP_SET_B:  case OP_SET_A:
699                 if (Wtautological_compare)
700                         warning(insn->pos, "self-comparison always evaluates to
701         case OP_SUB:
702         case OP_XOR:
703                 return replace_with_pseudo(insn, value_pseudo(insn->type, 0));

705         case OP_SET_EQ:
706         case OP_SET_LE: case OP_SET_GE:
707         case OP_SET_BE: case OP_SET_AE:
708                 if (Wtautological_compare)
709                         warning(insn->pos, "self-comparison always evaluates to
710                 return replace_with_pseudo(insn, value_pseudo(insn->type, 1));

712         case OP_AND:
713         case OP_OR:
714                 return replace_with_pseudo(insn, arg);

716         case OP_AND_BOOL:
717         case OP_OR_BOOL:
718                 remove_usage(arg, &insn->src2);
719                 insn->src2 = value_pseudo(insn->type, 0);
720                 insn->opcode = OP_SET_NE;
```



```
721                 return REPEAT_CSE;

723         default:
724                 break;
725         }

727         return 0;
728 }

730 static int simplify_binop(struct instruction *insn)
731 {
732         if (dead_insn(insn, &insn->src1, &insn->src2, NULL))
733                 return REPEAT_CSE;
734         if (constant(insn->src1)) {
735                 if (constant(insn->src2))
736                         return simplify_constant_binop(insn);
737                 return simplify_constant_leftside(insn);
738         }
739         if (constant(insn->src2))
740                 return simplify_constant_rightside(insn);
741         if (insn->src1 == insn->src2)
742                 return simplify_binop_same_args(insn, insn->src1);
743         return 0;
744 }

746 static void switch_pseudo(struct instruction *insn1, pseudo_t *pp1, struct instr
747 {
748         pseudo_t p1 = *pp1, p2 = *pp2;

750         use_pseudo(insn1, p2, pp1);
751         use_pseudo(insn2, p1, pp2);
752         remove_usage(p1, pp1);
753         remove_usage(p2, pp2);
754 }

756 static int canonical_order(pseudo_t p1, pseudo_t p2)
757 {
758         /* symbol/constants on the right */
759         if (p1->type == PSEUDO_VAL)
760                 return p2->type == PSEUDO_VAL;

762         if (p1->type == PSEUDO_SYM)
763                 return p2->type == PSEUDO_SYM || p2->type == PSEUDO_VAL;

765         return 1;
766 }

768 static int simplify_commutative_binop(struct instruction *insn)
769 {
770         if (!canonical_order(insn->src1, insn->src2)) {
771                 switch_pseudo(insn, &insn->src1, insn, &insn->src2);
772                 return REPEAT_CSE;
773         }
774         return 0;
775 }

777 static inline int simple_pseudo(pseudo_t pseudo)
778 {
779         return pseudo->type == PSEUDO_VAL || pseudo->type == PSEUDO_SYM;
780 }

782 static int simplify_associative_binop(struct instruction *insn)
783 {
784         struct instruction *def;
785         pseudo_t pseudo = insn->src1;
```

```
787         if (!simple_pseudo(insn->src2))
788                 return 0;
789         if (pseudo->type != PSEUDO_REG)
790                 return 0;
791         def = pseudo->def;
792         if (def == insn)
793                 return 0;
794         if (def->opcode != insn->opcode)
795                 return 0;
796         if (!simple_pseudo(def->src2))
797                 return 0;
798         if (ptr_list_size((struct ptr_list *)def->target->users) != 1)
799                 return 0;
800         switch_pseudo(def, &def->src1, insn, &insn->src2);
801         return REPEAT_CSE;
802 }

804 static int simplify_constant_unop(struct instruction *insn)
805 {
806         long long val = insn->src1->value;
807         long long res, mask;

809         switch (insn->opcode) {
810         case OP_NOT:
811                 res = ~val;
812                 break;
813         case OP_NEG:
814                 res = -val;
815                 break;
816         default:
817                 return 0;
818         }
819         mask = 1ULL << (insn->size-1);
820         res &= mask | (mask-1);
821
822         replace_with_pseudo(insn, value_pseudo(insn->type, res));
823         return REPEAT_CSE;
824 }

826 static int simplify_unop(struct instruction *insn)
827 {
828         if (dead_insn(insn, &insn->src1, NULL, NULL))
829                 return REPEAT_CSE;
830         if (constant(insn->src1))
831                 return simplify_constant_unop(insn);

833         switch (insn->opcode) {
834                 struct instruction *def;

836         case OP_NOT:
837                 def = insn->src->def;
838                 if (def && def->opcode == OP_NOT)
839                         return replace_with_pseudo(insn, def->src);
840                 break;
841         case OP_NEG:
842                 def = insn->src->def;
843                 if (def && def->opcode == OP_NEG)
844                         return replace_with_pseudo(insn, def->src);
845                 break;
846         default:
847                 return 0;
848         }
849         return 0;
850 }

852 static int simplify_one_memop(struct instruction *insn, pseudo_t orig)
```



```
853 {
854         pseudo_t addr = insn->src;
855         pseudo_t new, off;

857         if (addr->type == PSEUDO_REG) {
858                 struct instruction *def = addr->def;
859                 if (def->opcode == OP_SYMADDR && def->src) {
860                         kill_use(&insn->src);
861                         use_pseudo(insn, def->src, &insn->src);
862                         return REPEAT_CSE | REPEAT_SYMBOL_CLEANUP;
863                 }
864                 if (def->opcode == OP_ADD) {
865                         new = def->src1;
866                         off = def->src2;
867                         if (constant(off))
868                                 goto offset;
869                         new = off;
870                         off = def->src1;
871                         if (constant(off))
872                                 goto offset;
873                         return 0;
874                 }
875         }
876         return 0;

878 offset:
879         /* Invalid code */
880         if (new == orig) {
881                 if (new == VOID)
882                         return 0;
883                 /*
884                  * If some BB have been removed it is possible that this
885                  * memop is in fact part of a dead BB. In this case
886                  * we must not warn since nothing is wrong.
887                  * If not part of a dead BB this will be redone after
888                  * the BBs have been cleaned up.
889                  */
890                 if (repeat_phase & REPEAT_CFG_CLEANUP)
891                         return 0;
892                 new = VOID;
893                 warning(insn->pos, "crazy programmer");
894         }
895         insn->offset += off->value;
896         use_pseudo(insn, new, &insn->src);
897         remove_usage(addr, &insn->src);
898         return REPEAT_CSE | REPEAT_SYMBOL_CLEANUP;
899 }

901 /*
902  * We walk the whole chain of adds/subs backwards. That's not
903  * only more efficient, but it allows us to find loops.
904  */
905 static int simplify_memop(struct instruction *insn)
906 {
907         int one, ret = 0;
908         pseudo_t orig = insn->src;

910         do {
911                 one = simplify_one_memop(insn, orig);
912                 ret |= one;
913         } while (one);
914         return ret;
915 }

917 static long long get_cast_value(long long val, int old_size, int new_size, int s
918 {
```

```
919         long long mask;
920
921         if (sign && new_size > old_size) {
922                 mask = 1 << (old_size-1);
923                 if (val & mask)
924                         val |= ~(mask | (mask-1));
925         }
926         mask = 1 << (new_size-1);
927         return val & (mask | (mask-1));
928 }
929
930 static int simplify_cast(struct instruction *insn)
931 {
932         struct symbol *orig_type;
933         int orig_size, size;
934         pseudo_t src;
935
936         if (dead_insn(insn, &insn->src, NULL, NULL))
937                 return REPEAT_CSE;
938
939         orig_type = insn->orig_type;
940         if (!orig_type)
941                 return 0;
942
943         /* Keep casts with pointer on either side (not only case of OP_PTRCAST)
944         if (is_ptr_type(orig_type) || is_ptr_type(insn->type))
945                 return 0;
946
947         orig_size = orig_type->bit_size;
948         size = insn->size;
949         src = insn->src;
950
951         /* A cast of a constant? */
952         if (constant(src)) {
953                 int sign = orig_type->ctype.modifiers & MOD_SIGNED;
954                 long long val = get_cast_value(src->value, orig_size, size, sign
955                 src = value_pseudo(orig_type, val);
956                 goto simplify;
957         }
958
959         /* A cast of a "and" might be a no-op.. */
960         if (src->type == PSEUDO_REG) {
961                 struct instruction *def = src->def;
962                 if (def->opcode == OP_AND && def->size >= size) {
963                         pseudo_t val = def->src2;
964                         if (val->type == PSEUDO_VAL) {
965                                 unsigned long long value = val->value;
966                                 if (!(value >> (size-1)))
967                                         goto simplify;
968                         }
969                 }
970         }
971
972         if (size == orig_size) {
973                 int op = (orig_type->ctype.modifiers & MOD_SIGNED) ? OP_SCAST :
974                 if (insn->opcode == op)
975                         goto simplify;
976                 if (insn->opcode == OP_FPCAST && is_float_type(orig_type))
977                         goto simplify;
978         }
979
980         return 0;
981
982 simplify:
983         return replace_with_pseudo(insn, src);
984 }
```



```
985
986 static int simplify_select(struct instruction *insn)
987 {
988         pseudo_t cond, src1, src2;
989
990         if (dead_insn(insn, &insn->src1, &insn->src2, &insn->src3))
991                 return REPEAT_CSE;
992
993         cond = insn->src1;
994         src1 = insn->src2;
995         src2 = insn->src3;
996         if (constant(cond) || src1 == src2) {
997                 pseudo_t *kill, take;
998                 kill_use(&insn->src1);
999                 take = cond->value ? src1 : src2;
1000                kill = cond->value ? &insn->src3 : &insn->src2;
1001                kill_use(kill);
1002                replace_with_pseudo(insn, take);
1003                return REPEAT_CSE;
1004        }
1005        if (constant(src1) && constant(src2)) {
1006                long long val1 = src1->value;
1007                long long val2 = src2->value;
1008
1009                /* The pair 0/1 is special - replace with SETNE/SETEQ */
1010                if ((val1 | val2) == 1) {
1011                        int opcode = OP_SET_EQ;
1012                        if (val1) {
1013                                src1 = src2;
1014                                opcode = OP_SET_NE;
1015                        }
1016                        insn->opcode = opcode;
1017                        /* insn->src1 is already cond */
1018                        insn->src2 = src1; /* Zero */
1019                        return REPEAT_CSE;
1020                }
1021        }
1022        return 0;
1023 }
1024
1025 static int is_in_range(pseudo_t src, long long low, long long high)
1026 {
1027         long long value;
1028
1029         switch (src->type) {
1030         case PSEUDO_VAL:
1031                 value = src->value;
1032                 return value >= low && value <= high;
1033         default:
1034                 return 0;
1035         }
1036 }
1037
1038 static int simplify_range(struct instruction *insn)
1039 {
1040         pseudo_t src1, src2, src3;
1041
1042         src1 = insn->src1;
1043         src2 = insn->src2;
1044         src3 = insn->src3;
1045         if (src2->type != PSEUDO_VAL || src3->type != PSEUDO_VAL)
1046                 return 0;
1047         if (is_in_range(src1, src2->value, src3->value)) {
1048                 kill_instruction(insn);
1049                 return REPEAT_CSE;
1050         }
```

```
1051         return 0;
1052 }

1054 /*
1055  * Simplify "set_ne/eq $0 + br"
1056  */
1057 static int simplify_cond_branch(struct instruction *br, pseudo_t cond, struct in
1058 {
1059         use_pseudo(br, *pp, &br->cond);
1060         remove_usage(cond, &br->cond);
1061         if (def->opcode == OP_SET_EQ) {
1062                 struct basic_block *true = br->bb_true;
1063                 struct basic_block *false = br->bb_false;
1064                 br->bb_false = true;
1065                 br->bb_true = false;
1066         }
1067         return REPEAT_CSE;
1068 }

1070 static int simplify_branch(struct instruction *insn)
1071 {
1072         pseudo_t cond = insn->cond;

1074         /* Constant conditional */
1075         if (constant(cond)) {
1076                 insert_branch(insn->bb, insn, cond->value ? insn->bb_true : insn
1077                 return REPEAT_CSE;
1078         }

1080         /* Same target? */
1081         if (insn->bb_true == insn->bb_false) {
1082                 struct basic_block *bb = insn->bb;
1083                 struct basic_block *target = insn->bb_false;
1084                 remove_bb_from_list(&target->parents, bb, 1);
1085                 remove_bb_from_list(&bb->children, target, 1);
1086                 insn->bb_false = NULL;
1087                 kill_use(&insn->cond);
1088                 insn->cond = NULL;
1089                 insn->opcode = OP_BR;
1090                 return REPEAT_CSE;
1091         }

1093         /* Conditional on a SETNE $0 or SETEQ $0 */
1094         if (cond->type == PSEUDO_REG) {
1095                 struct instruction *def = cond->def;

1097                 if (def->opcode == OP_SET_NE || def->opcode == OP_SET_EQ) {
1098                         if (constant(def->src1) && !def->src1->value)
1099                                 return simplify_cond_branch(insn, cond, def, &de
1100                         if (constant(def->src2) && !def->src2->value)
1101                                 return simplify_cond_branch(insn, cond, def, &de
1102                 }
1103                 if (def->opcode == OP_SEL) {
1104                         if (constant(def->src2) && constant(def->src3)) {
1105                                 long long val1 = def->src2->value;
1106                                 long long val2 = def->src3->value;
1107                                 if (!val1 && !val2) {
1108                                         insert_branch(insn->bb, insn, insn->bb_f
1109                                         return REPEAT_CSE;
1110                                 }
1111                                 if (val1 && val2) {
1112                                         insert_branch(insn->bb, insn, insn->bb_t
1113                                         return REPEAT_CSE;
1114                                 }
1115                                 if (val2) {
1116                                         struct basic_block *true = insn->bb_true
```



```
1117                                         struct basic_block *false = insn->bb_fal
1118                                         insn->bb_false = true;
1119                                         insn->bb_true = false;
1120                                 }
1121                                 use_pseudo(insn, def->src1, &insn->cond);
1122                                 remove_usage(cond, &insn->cond);
1123                                 return REPEAT_CSE;
1124                         }
1125                 }
1126                 if (def->opcode == OP_CAST || def->opcode == OP_SCAST) {
1127                         int orig_size = def->orig_type ? def->orig_type->bit_siz
1128                         if (def->size > orig_size) {
1129                                 use_pseudo(insn, def->src, &insn->cond);
1130                                 remove_usage(cond, &insn->cond);
1131                                 return REPEAT_CSE;
1132                         }
1133                 }
1134         }
1135         return 0;
1136 }

1138 static int simplify_switch(struct instruction *insn)
1139 {
1140         pseudo_t cond = insn->cond;
1141         long long val;
1142         struct multijmp *jmp;

1144         if (!constant(cond))
1145                 return 0;
1146         val = insn->cond->value;

1148         FOR_EACH_PTR(insn->multijmp_list, jmp) {
1149                 /* Default case */
1150                 if (jmp->begin > jmp->end)
1151                         goto found;
1152                 if (val >= jmp->begin && val <= jmp->end)
1153                         goto found;
1154         } END_FOR_EACH_PTR(jmp);
1155         warning(insn->pos, "Impossible case statement");
1156         return 0;

1158 found:
1159         insert_branch(insn->bb, insn, jmp->target);
1160         return REPEAT_CSE;
1161 }

1163 int simplify_instruction(struct instruction *insn)
1164 {
1165         if (!insn->bb)
1166                 return 0;
1167         switch (insn->opcode) {
1168         case OP_ADD: case OP_MULS:
1169         case OP_AND: case OP_OR: case OP_XOR:
1170         case OP_AND_BOOL: case OP_OR_BOOL:
1171                 if (simplify_binop(insn))
1172                         return REPEAT_CSE;
1173                 if (simplify_commutative_binop(insn))
1174                         return REPEAT_CSE;
1175                 return simplify_associative_binop(insn);

1177         case OP_MULU:
1178         case OP_SET_EQ: case OP_SET_NE:
1179                 if (simplify_binop(insn))
1180                         return REPEAT_CSE;
1181                 return simplify_commutative_binop(insn);
```

```
1183         case OP_SUB:
1184         case OP_DIVU: case OP_DIVS:
1185         case OP_MODU: case OP_MODS:
1186         case OP_SHL:
1187         case OP_LSR: case OP_ASR:
1188         case OP_SET_LE: case OP_SET_GE:
1189         case OP_SET_LT: case OP_SET_GT:
1190         case OP_SET_B:  case OP_SET_A:
1191         case OP_SET_BE: case OP_SET_AE:
1192                 return simplify_binop(insn);

1194         case OP_NOT: case OP_NEG:
1195                 return simplify_unop(insn);
1196         case OP_LOAD: case OP_STORE:
1197                 return simplify_memop(insn);
1198         case OP_SYMADDR:
1199                 if (dead_insn(insn, NULL, NULL, NULL))
1200                         return REPEAT_CSE | REPEAT_SYMBOL_CLEANUP;
1201                 return replace_with_pseudo(insn, insn->symbol);
1202         case OP_CAST:
1203         case OP_SCAST:
1204         case OP_FPCAST:
1205         case OP_PTRCAST:
1206                 return simplify_cast(insn);
1207         case OP_PHI:
1208                 if (dead_insn(insn, NULL, NULL, NULL)) {
1209                         kill_use_list(insn->phi_list);
1210                         return REPEAT_CSE;
1211                 }
1212                 return clean_up_phi(insn);
1213         case OP_PHISOURCE:
1214                 if (dead_insn(insn, &insn->phi_src, NULL, NULL))
1215                         return REPEAT_CSE;
1216                 break;
1217         case OP_SEL:
1218                 return simplify_select(insn);
1219         case OP_CBR:
1220                 return simplify_branch(insn);
1221         case OP_SWITCH:
1222                 return simplify_switch(insn);
1223         case OP_RANGE:
1224                 return simplify_range(insn);
1225         }
1226         return 0;
1227 }
```

```
**********************************************************
    8472 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2006 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdio.h>
#include <unistd.h>
#include <libgen.h>
#include "smatch.h"
#include "check_list.h"

char *option_debug_check = (char *)"";
char *option_project_str = (char *)"smatch_generic";
static char *option_db_file = (char *)"smatch_db.sqlite";
enum project_type option_project = PROJ_NONE;
char *bin_dir;
char *data_dir;
int option_no_data = 0;
int option_spammy = 0;
int option_info = 0;
int option_full_path = 0;
int option_param_mapper = 0;
int option_call_tree = 0;
int option_no_db = 0;
int option_enable = 0;
int option_disable = 0;
int option_debug_related;
int option_file_output;
int option_time;
int option_mem;
char *option_datadir_str;
int option_fatal_checks;
int option_succeed;

FILE *sm_outfd;
FILE *sql_outfd;
FILE *caller_info_fd;

int sm_nr_errors;
int sm_nr_checks;

bool __silence_warnings_for_stmt;

const char *progname;

typedef void (*reg_func) (int id);
#define CK(_x) {.name = #_x, .func = &_x, .enabled = 0},
static struct reg_func_info {
```



```
        const char *name;
        reg_func func;
        int enabled;
} reg_funcs[] = {
        {NULL, NULL},
#include "check_list.h"
};
#undef CK
int num_checks = ARRAY_SIZE(reg_funcs) - 1;

const char *check_name(unsigned short id)
{
        if (id >= ARRAY_SIZE(reg_funcs))
                return "internal";

        return reg_funcs[id].name;
}

int id_from_name(const char *name)
{
        int i;

        for (i = 1; i < ARRAY_SIZE(reg_funcs); i++) {
                if (!strcmp(name, reg_funcs[i].name))
                        return i;
        }
        return 0;
}

static void show_checks(void)
{
        int i;

        for (i = 1; i < ARRAY_SIZE(reg_funcs); i++) {
                if (!strncmp(reg_funcs[i].name, "check_", 6))
                        printf("%3d. %s\n", i, reg_funcs[i].name);
        }
        exit(0);
}

static void enable_disable_checks(char *s, bool enable)
{
        char buf[128];
        char *next;
        int i;

        do {
                next = strchr(s, ',');
                if (next) {
                        *next = '\0';
                        next++;
                }
                if (*s == '\0')
                        return;
                if (strncmp(s, "check_", 6) == 0)
                        snprintf(buf, sizeof(buf), "%s", s);
                else
                        snprintf(buf, sizeof(buf), "check_%s", s);


                for (i = 1; i < ARRAY_SIZE(reg_funcs); i++) {
                        if (strcmp(reg_funcs[i].name, buf) == 0) {
                                reg_funcs[i].enabled = (enable == true) ? 1 : -1
                                break;
                        }
                }
```

```
128                 if (i == ARRAY_SIZE(reg_funcs))
129                         sm_fatal("'%s' not found", s);
130
131         } while ((s = next));
132 }
133
134 static void help(void)
135 {
136         printf("Usage:  smatch [smatch arguments][sparse arguments] file.c\n");
137         printf("--project=<name> or -p=<name>: project specific tests\n");
138         printf("--succeed: don't exit with an error\n");
139         printf("--spammy:  print superfluous crap.\n");
140         printf("--info:  print info used to fill smatch_data/.\n");
141         printf("--debug:  print lots of debug output.\n");
142         printf("--param-mapper:  enable param_mapper output.\n");
143         printf("--no-data:  do not use the /smatch_data/ directory.\n");
144         printf("--data=<dir>: overwrite path to default smatch data directory.\n
145         printf("--full-path:  print the full pathname.\n");
146         printf("--debug-implied:  print debug output about implications.\n");
147         printf("--assume-loops:  assume loops always go through at least once.\n
148         printf("--two-passes:  use a two pass system for each function.\n");
149         printf("--file-output:  instead of printing stdout, print to \"file.c.sm
150         printf("--fatal-checks: check output is treated as an error.\n");
151         printf("--help:  print this helpful message.\n");
152         exit(1);
153 }
154
155 static int match_option(const char *arg, const char *option)
156 {
157         char *str;
158         char *tmp;
159         int ret = 0;
160
161         str = malloc(strlen(option) + 3);
162         snprintf(str, strlen(option) + 3, "--%s", option);
163         tmp = str;
164         while (*tmp) {
165                 if (*tmp == '_')
166                         *tmp = '-';
167                 tmp++;
168         }
169         if (!strcmp(arg, str))
170                 ret = 1;
171         free(str);
172         return ret;
173 }
174
175 #define OPTION(_x) do {                                         \
176         if (match_option((*argvp)[1], #_x)) {                   \
177                 option_##_x = 1;                                \
178         }                                                       \
179 } while (0)
180
181 void parse_args(int *argcp, char ***argvp)
182 {
183         int i;
184
185         for (i = 1 ; i < *argcp; i++) {
186                 if (!strcmp((*argvp)[i], "--help"))
187                         help();
188
189                 if (!strcmp((*argvp)[i], "--show-checks"))
190                         show_checks();
191
192                 if (!strncmp((*argvp)[i], "--project=", 10))
```



```
193                         option_project_str = (*argvp)[i] + 10;
194
195                 if (!strncmp((*argvp)[i], "-p=", 3))
196                         option_project_str = (*argvp)[i] + 3;
197
198                 if (!strncmp((*argvp)[i], "--db-file=", 10))
199                         option_db_file = (*argvp)[i] + 10;
200
201                 if (!strncmp((*argvp)[i], "--data=", 7))
202                         option_datadir_str = (*argvp)[i] + 7;
203
204                 if (!strncmp((*argvp)[i], "--debug=", 8))
205                         option_debug_check = (*argvp)[i] + 8;
206
207                 if (strncmp((*argvp)[i], "--trace=", 8) == 0)
208                         trace_variable = (*argvp)[i] + 8;
209
210                 if (strncmp((*argvp)[i], "--enable=", 9) == 0) {
211                         enable_disable_checks((*argvp)[i] + 9, 1);
212                         option_enable = 1;
213                 }
214
215                 if (strncmp((*argvp)[i], "--disable=", 10) == 0) {
216                         enable_disable_checks((*argvp)[i] + 10, 0);
217                         option_enable = 1;
218                         option_disable = 1;
219                 }
220
221                 OPTION(fatal_checks);
222                 OPTION(spammy);
223                 OPTION(info);
224                 OPTION(debug);
225                 OPTION(debug_implied);
226                 OPTION(debug_related);
227                 OPTION(assume_loops);
228                 OPTION(no_data);
229                 OPTION(two_passes);
230                 OPTION(full_path);
231                 OPTION(param_mapper);
232                 OPTION(call_tree);
233                 OPTION(file_output);
234                 OPTION(time);
235                 OPTION(mem);
236                 OPTION(no_db);
237                 OPTION(succeed);
238         }
239
240         if (strcmp(option_project_str, "smatch_generic") != 0)
241                 option_project = PROJ_UNKNOWN;
242
243         if (strcmp(option_project_str, "kernel") == 0)
244                 option_project = PROJ_KERNEL;
245         else if (strcmp(option_project_str, "wine") == 0)
246                 option_project = PROJ_WINE;
247         else if (strcmp(option_project_str, "illumos_kernel") == 0)
248                 option_project = PROJ_ILLUMOS_KERNEL;
249         else if (strcmp(option_project_str, "illumos_user") == 0)
250                 option_project = PROJ_ILLUMOS_USER;
251 }
252
253 static char *read_bin_filename(void)
254 {
255         char filename[PATH_MAX] = {};
256         char proc[PATH_MAX];
257
258         pid_t pid = getpid();
```

```
259         sprintf(proc, "/proc/%d/exe", pid);
260         if (readlink(proc, filename, PATH_MAX) < 0)
261                 return NULL;
262         return alloc_string(filename);
263 }

265 static char *get_bin_dir(char *arg0)
266 {
267         char *orig;

269         orig = read_bin_filename();
270         if (!orig)
271                 orig = alloc_string(arg0);
272         return dirname(orig);
273 }

275 static char *get_data_dir(char *arg0)
276 {
277         char buf[256];
278         char *dir;

280         if (option_no_data)
281                 return NULL;

283         if (option_datadir_str) {
284                 if (access(option_datadir_str, R_OK))
285                         sm_warning("%s is not accessible -- ignored.",
286                                    option_datadir_str);
287                 else
288                         return alloc_string(option_datadir_str);
289         }

291         strncpy(buf, "smatch_data/", sizeof(buf));
292         dir = alloc_string(buf);
293         if (!access(dir, R_OK))
294                 return dir;

296         strncpy(buf, bin_dir, 254);

298         buf[255] = '\0';
299         strncat(buf, "/smatch_data/", 254 - strlen(buf));
300         dir = alloc_string(buf);
301         if (!access(dir, R_OK))
302                 return dir;
303         free_string(dir);
304         snprintf(buf, 254, "%s/smatch_data/", SMATCHDATADIR);
305         dir = alloc_string(buf);
306         if (!access(dir, R_OK))
307                 return dir;

309         sm_warning("%s is not accessible.", dir);
310         sm_warning("Use --no-data or --data to suppress this message.");
311         return NULL;
312 }

314 int main(int argc, char **argv)
315 {
316         int i;
317         reg_func func;

319         sm_outfd = stdout;
320         sql_outfd = stdout;
321         caller_info_fd = stdout;

323         progname = argv[0];
```



```
325         parse_args(&argc, &argv);

327         if (argc < 2)
328                 help();

330         /* this gets set back to zero when we parse the first function */
331         final_pass = 1;

333         bin_dir = get_bin_dir(argv[0]);
334         data_dir = get_data_dir(argv[0]);

336         allocate_hook_memory();
337         create_function_hook_hash();
338         open_smatch_db(option_db_file);
339         for (i = 1; i < ARRAY_SIZE(reg_funcs); i++) {
340                 func = reg_funcs[i].func;
341                 /* The script IDs start at 1.
342                    0 is used for internal stuff. */
343                 if (!option_enable || reg_funcs[i].enabled == 1 ||
344                     (option_disable && reg_funcs[i].enabled != -1) ||
345                     strncmp(reg_funcs[i].name, "register_", 9) == 0)
346                         func(i);
347         }

349         smatch(argc, argv);
350         free_string(data_dir);

352         if (option_succeed)
353                 return 0;
354         if (sm_nr_errors > 0)
355                 return 1;
356         if (sm_nr_checks > 0 && option_fatal_checks)
357                 return 1;
358         return 0;
359 }
```

```
**********************************************************
   45993 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2006 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#ifndef         SMATCH_H_
# define        SMATCH_H_

#include <stdio.h>
#include <string.h>
#include <limits.h>
#include <sys/time.h>
#include <sqlite3.h>
#include "lib.h"
#include "allocate.h"
#include "scope.h"
#include "parse.h"
#include "expression.h"
#include "avl.h"

typedef struct {
        struct symbol *type;
        union {
                long long value;
                unsigned long long uvalue;
        };
} sval_t;

typedef long long mtag_t;

struct smatch_state {
        const char *name;
        void *data;
};
#define STATE(_x) static struct smatch_state _x = { .name = #_x }
extern struct smatch_state undefined;
extern struct smatch_state ghost;
extern struct smatch_state merged;
extern struct smatch_state true_state;
extern struct smatch_state false_state;
DECLARE_ALLOCATOR(smatch_state);

static inline void *INT_PTR(int i)
{
        return (void *)(long)i;
}

static inline int PTR_INT(void *p)
```



```
{
        return (int)(long)p;
}

struct tracker {
        char *name;
        struct symbol *sym;
        unsigned short owner;
};
DECLARE_ALLOCATOR(tracker);
DECLARE_PTR_LIST(tracker_list, struct tracker);
DECLARE_PTR_LIST(stree_stack, struct stree);

/* The first 3 struct members must match struct tracker */
struct sm_state {
        const char *name;
        struct symbol *sym;
        unsigned short owner;
        unsigned short merged:1;
        unsigned short skip_implications:1;
        unsigned int nr_children;
        unsigned int line;
        struct smatch_state *state;
        struct stree *pool;
        struct sm_state *left;
        struct sm_state *right;
        struct state_list *possible;
};

struct var_sym {
        char *var;
        struct symbol *sym;
};
DECLARE_ALLOCATOR(var_sym);
DECLARE_PTR_LIST(var_sym_list, struct var_sym);

struct constraint {
        int op;
        int id;
};
DECLARE_PTR_LIST(constraint_list, struct constraint);

enum hook_type {
        EXPR_HOOK,
        STMT_HOOK,
        STMT_HOOK_AFTER,
        SYM_HOOK,
        STRING_HOOK,
        DECLARATION_HOOK,
        ASSIGNMENT_HOOK,
        ASSIGNMENT_HOOK_AFTER,
        RAW_ASSIGNMENT_HOOK,
        GLOBAL_ASSIGNMENT_HOOK,
        LOGIC_HOOK,
        CONDITION_HOOK,
        PRELOOP_HOOK,
        SELECT_HOOK,
        WHOLE_CONDITION_HOOK,
        FUNCTION_CALL_HOOK_BEFORE,
        FUNCTION_CALL_HOOK,
        CALL_HOOK_AFTER_INLINE,
        FUNCTION_CALL_HOOK_AFTER_DB,
        CALL_ASSIGNMENT_HOOK,
        MACRO_ASSIGNMENT_HOOK,
        BINOP_HOOK,
        OP_HOOK,
```

```
127         DEREF_HOOK,
128         CASE_HOOK,
129         ASM_HOOK,
130         CAST_HOOK,
131         SIZEOF_HOOK,
132         BASE_HOOK,
133         FUNC_DEF_HOOK,
134         AFTER_DEF_HOOK,
135         END_FUNC_HOOK,
136         AFTER_FUNC_HOOK,
137         RETURN_HOOK,
138         INLINE_FN_START,
139         INLINE_FN_END,
140         END_FILE_HOOK,
141         NUM_HOOKS,
142 };

144 #define TRUE 1
145 #define FALSE 0

147 struct range_list;

149 void add_hook(void *func, enum hook_type type);
150 typedef struct smatch_state *(merge_func_t)(struct smatch_state *s1, struct smat
151 typedef struct smatch_state *(unmatched_func_t)(struct sm_state *state);
152 void add_merge_hook(int client_id, merge_func_t *func);
153 void add_unmatched_state_hook(int client_id, unmatched_func_t *func);
154 void add_pre_merge_hook(int client_id, void (*hook)(struct sm_state *sm));
155 typedef void (scope_hook)(void *data);
156 void add_scope_hook(scope_hook *hook, void *data);
157 typedef void (func_hook)(const char *fn, struct expression *expr, void *data);
158 typedef void (implication_hook)(const char *fn, struct expression *call_expr,
159                                 struct expression *assign_expr, void *data);
160 typedef void (return_implies_hook)(struct expression *call_expr,
161                                    int param, char *key, char *value);
162 typedef int (implied_return_hook)(struct expression *call_expr, void *info, stru
163 void add_function_hook(const char *look_for, func_hook *call_back, void *data);

165 void add_function_assign_hook(const char *look_for, func_hook *call_back,
166                               void *info);
167 void add_implied_return_hook(const char *look_for,
168                              implied_return_hook *call_back,
169                              void *info);
170 void add_macro_assign_hook(const char *look_for, func_hook *call_back,
171                               void *info);
172 void add_macro_assign_hook_extra(const char *look_for, func_hook *call_back,
173                               void *info);
174 void return_implies_state(const char *look_for, long long start, long long end,
175                           implication_hook *call_back, void *info);
176 void select_return_states_hook(int type, return_implies_hook *callback);
177 void select_return_states_before(void (*fn)(void));
178 void select_return_states_after(void (*fn)(void));
179 int get_implied_return(struct expression *expr, struct range_list **rl);
180 void allocate_hook_memory(void);

182 struct modification_data {
183         struct smatch_state *prev;
184         struct expression *cur;
185 };

187 typedef void (modification_hook)(struct sm_state *sm, struct expression *mod_exp
188 void add_modification_hook(int owner, modification_hook *call_back);
189 void add_modification_hook_late(int owner, modification_hook *call_back);
190 struct smatch_state *get_modification_state(struct expression *expr);

192 int outside_of_function(void);
```



```
193 const char *get_filename(void);
194 const char *get_base_file(void);
195 char *get_function(void);
196 int get_lineno(void);
197 extern int final_pass;
198 extern struct symbol *cur_func_sym;
199 extern int option_debug;
200 extern int local_debug;
201 extern int option_info;
202 extern int option_spammy;
203 extern char *trace_variable;
204 extern struct stree *global_states;
205 int is_skipped_function(void);
206 int is_silenced_function(void);

208 /* smatch_impossible.c */
209 int is_impossible_path(void);
210 void set_path_impossible(void);

212 extern FILE *sm_outfd;
213 extern FILE *sql_outfd;
214 extern FILE *caller_info_fd;
215 extern int sm_nr_checks;
216 extern int sm_nr_errors;
217 extern const char *progname;

219 /*
220  * How to use these routines:
221  *
222  * sm_fatal(): an internal error of some kind that should immediately exit
223  * sm_ierror(): an internal error
224  * sm_perror(): an internal error from parsing input source
225  * sm_error(): an error from input source
226  * sm_warning(): a warning from input source
227  * sm_info(): info message (from option_info)
228  * sm_debug(): debug message
229  * sm_msg(): other message (please avoid using this)
230  */

232 #define sm_printf(msg...) do { if (final_pass || option_debug || local_debug) fp

234 static inline void sm_prefix(void)
235 {
236         sm_printf("%s: %s:%d %s() ", progname, get_filename(), get_lineno(), get
237 }

239 static inline void print_implied_debug_msg();

241 extern bool __silence_warnings_for_stmt;

243 #define sm_print_msg(type, msg...) \
244 do {                                                            \
245         print_implied_debug_msg();                              \
246         if (!final_pass && !option_debug && !local_debug)       \
247                 break;                                          \
248         if (__silence_warnings_for_stmt && !option_debug && !local_debug) \
249                 break;                                          \
250         if (!option_info && is_silenced_function())             \
251                 break;                                          \
252         sm_prefix();                                            \
253         if (type == 1) {                                        \
254                 sm_printf("warn: ");                           \
255                 sm_nr_checks++;                                 \
256         } else if (type == 2) {                                 \
257                 sm_printf("error: ");                          \
258                 sm_nr_checks++;                                 \
```

```
259         } else if (type == 3) {                                     \
260                 sm_printf("parse error: ");                         \
261                 sm_nr_errors++;                                     \
262         }                                                           \
263         sm_printf(msg);                                             \
264         sm_printf("\n");                                           \
265 } while (0)

267 #define sm_msg(msg...) do { sm_print_msg(0, msg); } while (0)

269 #define local_debug(msg...)                                         \
270 do {                                                                \
271         if (local_debug)                                            \
272                 sm_msg(msg);                                        \
273 } while (0)

275 extern char *implied_debug_msg;
276 static inline void print_implied_debug_msg(void)
277 {
278         static struct symbol *last_printed = NULL;

280         if (!implied_debug_msg)
281                 return;
282         if (last_printed == cur_func_sym)
283                 return;
284         last_printed = cur_func_sym;
285         sm_msg("%s", implied_debug_msg);
286 }

288 #define sm_debug(msg...) do { if (option_debug) sm_printf(msg); } while (0)

290 #define sm_info(msg...) do {                                        \
291         if (option_debug || (option_info && final_pass)) {         \
292                 sm_prefix();                                        \
293                 sm_printf("info: ");                               \
294                 sm_printf(msg);                                     \
295                 sm_printf("\n");                                    \
296         }                                                           \
297 } while(0)

299 #define sm_warning(msg...) do { sm_print_msg(1, msg); } while (0)
300 #define sm_error(msg...) do { sm_print_msg(2, msg); } while (0)
301 #define sm_perror(msg...) do { sm_print_msg(3, msg); } while (0)

303 static inline void sm_fatal(const char *fmt, ...)
304 {
305         va_list args;

307         va_start(args, fmt);
308         vfprintf(sm_outfd, fmt, args);
309         va_end(args);

311         fprintf(sm_outfd, "\n");

313         exit(1);
314 }

316 static inline void sm_ierror(const char *fmt, ...)
317 {
318         va_list args;

320         sm_nr_errors++;

322         fprintf(sm_outfd, "internal error: ");

324         va_start(args, fmt);
```



```
325         vfprintf(sm_outfd, fmt, args);
326         va_end(args);

328         fprintf(sm_outfd, "\n");
329 }
330 #define ALIGN(x, a) (((x) + (a) - 1) & ~((a) - 1))

332 struct smatch_state *__get_state(int owner, const char *name, struct symbol *sym
333 struct smatch_state *get_state(int owner, const char *name, struct symbol *sym);
334 struct smatch_state *get_state_expr(int owner, struct expression *expr);
335 struct state_list *get_possible_states(int owner, const char *name,
336                                        struct symbol *sym);
337 struct state_list *get_possible_states_expr(int owner, struct expression *expr);
338 struct sm_state *set_state(int owner, const char *name, struct symbol *sym,
339                 struct smatch_state *state);
340 struct sm_state *set_state_expr(int owner, struct expression *expr,
341                 struct smatch_state *state);
342 void delete_state(int owner, const char *name, struct symbol *sym);
343 void delete_state_expr(int owner, struct expression *expr);
344 void __delete_all_states_sym(struct symbol *sym);
345 void set_true_false_states(int owner, const char *name, struct symbol *sym,
346                            struct smatch_state *true_state,
347                            struct smatch_state *false_state);
348 void set_true_false_states_expr(int owner, struct expression *expr,
349                            struct smatch_state *true_state,
350                            struct smatch_state *false_state);

352 struct stree *get_all_states_from_stree(int owner, struct stree *source);
353 struct stree *get_all_states_stree(int id);
354 struct stree *__get_cur_stree(void);
355 int is_reachable(void);
356 void add_get_state_hook(void (*fn)(int owner, const char *name, struct symbol *s

358 /* smatch_helper.c */
359 DECLARE_PTR_LIST(int_stack, int);
360 char *alloc_string(const char *str);
361 void free_string(char *str);
362 void append(char *dest, const char *data, int buff_len);
363 void remove_parens(char *str);
364 struct smatch_state *alloc_state_num(int num);
365 struct smatch_state *alloc_state_str(const char *name);
366 struct smatch_state *alloc_state_expr(struct expression *expr);
367 struct expression *get_argument_from_call_expr(struct expression_list *args,
368                                                int num);

370 char *expr_to_var(struct expression *expr);
371 struct symbol *expr_to_sym(struct expression *expr);
372 char *expr_to_str(struct expression *expr);
373 char *expr_to_str_sym(struct expression *expr,
374                                       struct symbol **sym_ptr);
375 char *expr_to_var_sym(struct expression *expr,
376                                  struct symbol **sym_ptr);
377 char *expr_to_known_chunk_sym(struct expression *expr, struct symbol **sym);
378 char *expr_to_chunk_sym_vsl(struct expression *expr, struct symbol **sym, struct
379 int get_complication_score(struct expression *expr);

381 int sym_name_is(const char *name, struct expression *expr);
382 int get_const_value(struct expression *expr, sval_t *sval);
383 int get_value(struct expression *expr, sval_t *val);
384 int get_implied_value(struct expression *expr, sval_t *val);
385 int get_implied_min(struct expression *expr, sval_t *sval);
386 int get_implied_max(struct expression *expr, sval_t *val);
387 int get_hard_max(struct expression *expr, sval_t *sval);
388 int get_fuzzy_min(struct expression *expr, sval_t *min);
389 int get_fuzzy_max(struct expression *expr, sval_t *max);
390 int get_absolute_min(struct expression *expr, sval_t *sval);
```

```
391 int get_absolute_max(struct expression *expr, sval_t *sval);
392 int parse_call_math(struct expression *expr, char *math, sval_t *val);
393 int parse_call_math_rl(struct expression *call, char *math, struct range_list **
394 char *get_value_in_terms_of_parameter_math(struct expression *expr);
395 char *get_value_in_terms_of_parameter_math_var_sym(const char *var, struct symbo
396 int is_zero(struct expression *expr);
397 int known_condition_true(struct expression *expr);
398 int known_condition_false(struct expression *expr);
399 int implied_condition_true(struct expression *expr);
400 int implied_condition_false(struct expression *expr);
401 int can_integer_overflow(struct symbol *type, struct expression *expr);
402 void clear_math_cache(void);

404 int is_array(struct expression *expr);
405 struct expression *get_array_base(struct expression *expr);
406 struct expression *get_array_offset(struct expression *expr);
407 const char *show_state(struct smatch_state *state);
408 struct statement *get_expression_statement(struct expression *expr);
409 struct expression *strip_parens(struct expression *expr);
410 struct expression *strip_expr(struct expression *expr);
411 struct expression *strip_expr_set_parent(struct expression *expr);
412 void scoped_state(int my_id, const char *name, struct symbol *sym);
413 int is_error_return(struct expression *expr);
414 int getting_address(void);
415 int get_struct_and_member(struct expression *expr, const char **type, const char
416 char *get_member_name(struct expression *expr);
417 char *get_fnptr_name(struct expression *expr);
418 int cmp_pos(struct position pos1, struct position pos2);
419 int positions_eq(struct position pos1, struct position pos2);
420 struct statement *get_current_statement(void);
421 struct statement *get_prev_statement(void);
422 struct expression *get_last_expr_from_expression_stmt(struct expression *expr);
423 int get_param_num_from_sym(struct symbol *sym);
424 int get_param_num(struct expression *expr);
425 int ms_since(struct timeval *start);
426 int parent_is_gone_var_sym(const char *name, struct symbol *sym);
427 int parent_is_gone(struct expression *expr);
428 int invert_op(int op);
429 int expr_equiv(struct expression *one, struct expression *two);
430 void push_int(struct int_stack **stack, int num);
431 int pop_int(struct int_stack **stack);

433 /* smatch_type.c */
434 struct symbol *get_real_base_type(struct symbol *sym);
435 int type_bytes(struct symbol *type);
436 int array_bytes(struct symbol *type);
437 struct symbol *get_pointer_type(struct expression *expr);
438 struct symbol *get_type(struct expression *expr);
439 struct symbol *get_final_type(struct expression *expr);
440 struct symbol *get_promoted_type(struct symbol *left, struct symbol *right);
441 int type_signed(struct symbol *base_type);
442 int expr_unsigned(struct expression *expr);
443 int expr_signed(struct expression *expr);
444 int returns_unsigned(struct symbol *base_type);
445 int is_pointer(struct expression *expr);
446 int returns_pointer(struct symbol *base_type);
447 sval_t sval_type_max(struct symbol *base_type);
448 sval_t sval_type_min(struct symbol *base_type);
449 int nr_bits(struct expression *expr);
450 int is_void_pointer(struct expression *expr);
451 int is_char_pointer(struct expression *expr);
452 int is_string(struct expression *expr);
453 int is_static(struct expression *expr);
454 int is_local_variable(struct expression *expr);
455 int types_equiv(struct symbol *one, struct symbol *two);
456 int fn_static(void);
```



```
457 const char *global_static();
458 struct symbol *cur_func_return_type(void);
459 struct symbol *get_arg_type(struct expression *fn, int arg);
460 struct symbol *get_member_type_from_key(struct expression *expr, const char *key
461 struct symbol *get_arg_type_from_key(struct expression *fn, int param, struct ex
462 int is_struct(struct expression *expr);
463 char *type_to_str(struct symbol *type);

465 /* smatch_ignore.c */
466 void add_ignore(int owner, const char *name, struct symbol *sym);
467 int is_ignored(int owner, const char *name, struct symbol *sym);
468 void add_ignore_expr(int owner, struct expression *expr);
469 int is_ignored_expr(int owner, struct expression *expr);

471 /* smatch_var_sym */
472 struct var_sym *alloc_var_sym(const char *var, struct symbol *sym);
473 struct var_sym_list *expr_to_vsl(struct expression *expr);
474 void add_var_sym(struct var_sym_list **list, const char *var, struct symbol *sym
475 void add_var_sym_expr(struct var_sym_list **list, struct expression *expr);
476 void del_var_sym(struct var_sym_list **list, const char *var, struct symbol *sym
477 int in_var_sym_list(struct var_sym_list *list, const char *var, struct symbol *s
478 struct var_sym_list *clone_var_sym_list(struct var_sym_list *from_vsl);
479 void merge_var_sym_list(struct var_sym_list **dest, struct var_sym_list *src);
480 struct var_sym_list *combine_var_sym_lists(struct var_sym_list *one, struct var_
481 int var_sym_lists_equiv(struct var_sym_list *one, struct var_sym_list *two);
482 void free_var_sym_list(struct var_sym_list **list);
483 void free_var_syms_and_list(struct var_sym_list **list);

485 /* smatch_tracker */
486 struct tracker *alloc_tracker(int owner, const char *name, struct symbol *sym);
487 void add_tracker(struct tracker_list **list, int owner, const char *name,
488                  struct symbol *sym);
489 void add_tracker_expr(struct tracker_list **list, int owner, struct expression *
490 void del_tracker(struct tracker_list **list, int owner, const char *name,
491                  struct symbol *sym);
492 int in_tracker_list(struct tracker_list *list, int owner, const char *name,
493                  struct symbol *sym);
494 void free_tracker_list(struct tracker_list **list);
495 void free_trackers_and_list(struct tracker_list **list);

497 /* smatch_conditions */
498 int in_condition(void);

500 /* smatch_flow.c */

502 extern int __in_fake_assign;
503 extern int __in_fake_parameter_assign;
504 extern int __in_fake_struct_assign;
505 extern int in_fake_env;
506 void smatch (int argc, char **argv);
507 int inside_loop(void);
508 int definitely_inside_loop(void);
509 struct expression *get_switch_expr(void);
510 int in_expression_statement(void);
511 void __process_post_op_stack(void);
512 void __split_expr(struct expression *expr);
513 void __split_label_stmt(struct statement *stmt);
514 void __split_stmt(struct statement *stmt);
515 extern int __in_function_def;
516 extern int option_assume_loops;
517 extern int option_two_passes;
518 extern int option_no_db;
519 extern int option_file_output;
520 extern int option_time;
521 extern struct expression_list *big_expression_stack;
522 extern struct expression_list *big_condition_stack;
```

```
523 extern struct statement_list *big_statement_stack;
524 int is_assigned_call(struct expression *expr);
525 int inlinable(struct expression *expr);
526 extern int __inline_call;
527 extern struct expression *__inline_fn;
528 extern int __in_pre_condition;
529 extern int __bail_on_rest_of_function;
530 extern struct statement *__prev_stmt;
531 extern struct statement *__cur_stmt;
532 extern struct statement *__next_stmt;
533 void init_fake_env(void);
534 void end_fake_env(void);
535 int time_parsing_function(void);

537 /* smatch_struct_assignment.c */
538 struct expression *get_faked_expression(void);
539 void __fake_struct_member_assignments(struct expression *expr);

541 /* smatch_project.c */
542 int is_no_inline_function(const char *function);

544 /* smatch_conditions */
545 void __split_whole_condition(struct expression *expr);
546 void __handle_logic(struct expression *expr);
547 int is_condition(struct expression *expr);
548 int __handle_condition_assigns(struct expression *expr);
549 int __handle_select_assigns(struct expression *expr);
550 int __handle_expr_statement_assigns(struct expression *expr);

552 /* smatch_implied.c */
553 extern int option_debug_implied;
554 extern int option_debug_related;
555 struct range_list_stack;
556 void param_limit_implications(struct expression *expr, int param, char *key, cha
557 struct stree *__implied_case_stree(struct expression *switch_expr,
558                                    struct range_list *case_rl,
559                                    struct range_list_stack **remaining_cases,
560                                    struct stree **raw_stree);
561 void overwrite_states_using_pool(struct sm_state *gate_sm, struct sm_state *pool
562 int assume(struct expression *expr);
563 void end_assume(void);
564 int impossible_assumption(struct expression *left, int op, sval_t sval);

566 /* smatch_extras.c */
567 #define SMATCH_EXTRA 5 /* this is my_id from smatch extra set in smatch.c */
568 extern int RETURN_ID;

570 struct data_range {
571         sval_t min;
572         sval_t max;
573 };

575 #define MTAG_ALIAS_BIT (1ULL << 63)
576 #define MTAG_OFFSET_MASK 0xfffULL

578 extern long long valid_ptr_min, valid_ptr_max;
579 extern sval_t valid_ptr_min_sval, valid_ptr_max_sval;
580 extern struct range_list *valid_ptr_rl;
581 static const sval_t array_min_sval = {
582         .type = &ptr_ctype,
583         {.value = 100000},
584 };
585 static const sval_t array_max_sval = {
586         .type = &ptr_ctype,
587         {.value = 199999},
588 };
```



```
589 static const sval_t text_seg_min = {
590         .type = &ptr_ctype,
591         {.value = 100000000},
592 };
593 static const sval_t text_seg_max = {
594         .type = &ptr_ctype,
595         {.value = 177777777},
596 };
597 static const sval_t data_seg_min = {
598         .type = &ptr_ctype,
599         {.value = 200000000},
600 };
601 static const sval_t data_seg_max = {
602         .type = &ptr_ctype,
603         {.value = 277777777},
604 };
605 static const sval_t bss_seg_min = {
606         .type = &ptr_ctype,
607         {.value = 300000000},
608 };
609 static const sval_t bss_seg_max = {
610         .type = &ptr_ctype,
611         {.value = 377777777},
612 };
613 static const sval_t stack_seg_min = {
614         .type = &ptr_ctype,
615         {.value = 400000000},
616 };
617 static const sval_t stack_seg_max = {
618         .type = &ptr_ctype,
619         {.value = 477777777},
620 };
621 static const sval_t kmalloc_seg_min = {
622         .type = &ptr_ctype,
623         {.value = 500000000},
624 };
625 static const sval_t kmalloc_seg_max = {
626         .type = &ptr_ctype,
627         {.value = 577777777},
628 };
629 static const sval_t vmalloc_seg_min = {
630         .type = &ptr_ctype,
631         {.value = 600000000},
632 };
633 static const sval_t vmalloc_seg_max = {
634         .type = &ptr_ctype,
635         {.value = 677777777},
636 };
637 static const sval_t fn_ptr_min = {
638         .type = &ptr_ctype,
639         {.value = 700000000},
640 };
641 static const sval_t fn_ptr_max = {
642         .type = &ptr_ctype,
643         {.value = 777777777},
644 };

646 char *get_other_name_sym(const char *name, struct symbol *sym, struct symbol **n
647 char *map_call_to_other_name_sym(const char *name, struct symbol *sym, struct sy
648 char *map_long_to_short_name_sym(const char *name, struct symbol *sym, struct sy
649 char *map_long_to_short_name_sym_nostack(const char *name, struct symbol *sym, s

651 #define STRLEN_MAX_RET 1010101

653 /* smatch_absolute.c */
654 int get_absolute_min_helper(struct expression *expr, sval_t *sval);
```

```
655 int get_absolute_max_helper(struct expression *expr, sval_t *sval);

657 /* smatch_local_values.c */
658 int get_local_rl(struct expression *expr, struct range_list **rl);
659 int get_local_max_helper(struct expression *expr, sval_t *sval);
660 int get_local_min_helper(struct expression *expr, sval_t *sval);

662 /* smatch_type_value.c */
663 int get_db_type_rl(struct expression *expr, struct range_list **rl);
664 /* smatch_data_val.c */
665 int get_mtag_rl(struct expression *expr, struct range_list **rl);
666 /* smatch_array_values.c */
667 int get_array_rl(struct expression *expr, struct range_list **rl);

669 /* smatch_states.c */
670 void __swap_cur_stree(struct stree *stree);
671 void __push_fake_cur_stree();
672 struct stree *__pop_fake_cur_stree();
673 void __free_fake_cur_stree();
674 void __set_fake_cur_stree_fast(struct stree *stree);
675 void __pop_fake_cur_stree_fast(void);
676 void __merge_stree_into_cur(struct stree *stree);

678 int unreachable(void);
679 void __set_sm(struct sm_state *sm);
680 void __set_sm_cur_stree(struct sm_state *sm);
681 void __set_sm_fake_stree(struct sm_state *sm);
682 void __set_true_false_sm(struct sm_state *true_state,
683                          struct sm_state *false_state);
684 void nullify_path(void);
685 void __match_nullify_path_hook(const char *fn, struct expression *expr,
686                                void *unused);
687 void __unnullify_path(void);
688 int __path_is_null(void);
689 void save_all_states(void);
690 void restore_all_states(void);
691 void free_goto_stack(void);
692 void clear_all_states(void);

694 struct sm_state *get_sm_state(int owner, const char *name,
695                               struct symbol *sym);
696 struct sm_state *get_sm_state_expr(int owner, struct expression *expr);
697 void __push_true_states(void);
698 void __use_false_states(void);
699 void __discard_false_states(void);
700 void __merge_false_states(void);
701 void __merge_true_states(void);

703 void __negate_cond_stacks(void);
704 void __use_pre_cond_states(void);
705 void __use_cond_true_states(void);
706 void __use_cond_false_states(void);
707 void __push_cond_stacks(void);
708 void __fold_in_set_states(void);
709 void __free_set_states(void);
710 struct stree *__copy_cond_true_states(void);
711 struct stree *__copy_cond_false_states(void);
712 struct stree *__pop_cond_true_stack(void);
713 struct stree *__pop_cond_false_stack(void);
714 void __and_cond_states(void);
715 void __or_cond_states(void);
716 void __save_pre_cond_states(void);
717 void __discard_pre_cond_states(void);
718 struct stree *__get_true_states(void);
719 struct stree *__get_false_states(void);
720 void __use_cond_states(void);
```



```
721 extern struct state_list *__last_base_slist;

723 void __push_continues(void);
724 void __discard_continues(void);
725 void __process_continues(void);
726 void __merge_continues(void);

728 void __push_breaks(void);
729 void __process_breaks(void);
730 int __has_breaks(void);
731 void __merge_breaks(void);
732 void __use_breaks(void);

734 void __save_switch_states(struct expression *switch_expr);
735 void __discard_switches(void);
736 int have_remaining_cases(void);
737 void __merge_switches(struct expression *switch_expr, struct range_list *case_rl
738 void __push_default(void);
739 void __set_default(void);
740 int __pop_default(void);

742 void __push_conditions(void);
743 void __discard_conditions(void);

745 void __save_gotos(const char *name, struct symbol *sym);
746 void __merge_gotos(const char *name, struct symbol *sym);

748 void __print_cur_stree(void);

750 /* smatch_hooks.c */
751 void __pass_to_client(void *data, enum hook_type type);
752 void __pass_to_client_no_data(enum hook_type type);
753 void __pass_case_to_client(struct expression *switch_expr,
754                            struct range_list *rl);
755 int __has_merge_function(int client_id);
756 struct smatch_state *__client_merge_function(int owner,
757                                              struct smatch_state *s1,
758                                              struct smatch_state *s2);
759 struct smatch_state *__client_unmatched_state_function(struct sm_state *sm);
760 void call_pre_merge_hook(struct sm_state *sm);
761 void __push_scope_hooks(void);
762 void __call_scope_hooks(void);

764 /* smatch_function_hooks.c */
765 void create_function_hook_hash(void);
766 void __match_initializer_call(struct symbol *sym);

768 /* smatch_db.c */
769 enum info_type {
770         INTERNAL        = 0,
771         /*
772          * Changing these numbers is a pain.  Don't do it.  If you ever use a
773          * number it can't be re-used right away so there may be gaps.
774          * We select these in order by type so if the order matters, then give
775          * it a number below 100-999,9000-9999 ranges. */

777         PARAM_CLEARED   = 101,
778         PARAM_LIMIT     = 103,
779         PARAM_FILTER    = 104,

781         PARAM_VALUE     = 1001,
782         BUF_SIZE        = 1002,
783         USER_DATA       = 1003,
784         CAPPED_DATA     = 1004,
785         RETURN_VALUE    = 1005,
786         DEREFERENCE     = 1006,
```

```
787         RANGE_CAP       = 1007,
788         LOCK_HELD       = 1008,
789         LOCK_RELEASED   = 1009,
790         ABSOLUTE_LIMITS = 1010,
791         PARAM_ADD       = 1012,
792         PARAM_FREED     = 1013,
793         DATA_SOURCE     = 1014,
794         FUZZY_MAX       = 1015,
795         STR_LEN         = 1016,
796         ARRAY_LEN       = 1017,
797         CAPABLE         = 1018,
798         NS_CAPABLE      = 1019,
799         CONTAINER       = 1020,
800         CASTED_CALL     = 1021,
801         TYPE_LINK       = 1022,
802         UNTRACKED_PARAM = 1023,
803         CULL_PATH       = 1024,
804         PARAM_SET       = 1025,
805         PARAM_USED      = 1026,
806         BYTE_UNITS      = 1027,
807         COMPARE_LIMIT   = 1028,
808         PARAM_COMPARE   = 1029,
809         CONSTRAINT      = 1031,
810         PASSES_TYPE     = 1032,
811         CONSTRAINT_REQUIRED = 1033,
812         NOSPEC          = 1035,
813         NOSPEC_WB       = 1036,
814         STMT_CNT        = 1037,
815         TERMINATED      = 1038,

817         /* put random temporary stuff in the 7000-7999 range for testing */
818         USER_DATA3      = 8017,
819         USER_DATA3_SET  = 9017,
820         NO_OVERFLOW     = 8018,
821         NO_OVERFLOW_SIMPLE = 8019,
822         LOCKED          = 8020,
823         UNLOCKED        = 8021,
824         SET_FS          = 8022,
825         ATOMIC_INC      = 8023,
826         ATOMIC_DEC      = 8024,
827         NO_SIDE_EFFECT  = 8025,
828         FN_ARG_LINK     = 8028,
829         DATA_VALUE      = 8029,
830         ARRAYSIZE_ARG   = 8033,
831         SIZEOF_ARG      = 8034,
832         MEMORY_TAG      = 8036,
833         MTAG_ASSIGN     = 8035,
834         STRING_VALUE    = 8041,
835 };

837 extern struct sqlite3 *smatch_db;
838 extern struct sqlite3 *mem_db;
839 extern struct sqlite3 *cache_db;

841 void db_ignore_states(int id);
842 void select_caller_info_hook(void (*callback)(const char *name, struct symbol *s
843 void add_member_info_callback(int owner, void (*callback)(struct expression *cal
844 void add_split_return_callback(void (*fn)(int return_id, char *return_ranges, st
845 void add_returned_member_callback(int owner, void (*callback)(int return_id, cha
846 void select_call_implies_hook(int type, void (*callback)(struct expression *call
847 void select_return_implies_hook(int type, void (*callback)(struct expression *ca
848 struct range_list *db_return_vals(struct expression *expr);
849 struct range_list *db_return_vals_from_str(const char *fn_name);
850 char *return_state_to_var_sym(struct expression *expr, int param, const char *ke
851 char *get_chunk_from_key(struct expression *arg, char *key, struct symbol **sym,
852 char *get_variable_from_key(struct expression *arg, const char *key, struct symb
```



```
853 const char *state_name_to_param_name(const char *state_name, const char *param_n
854 const char *get_param_name_var_sym(const char *name, struct symbol *sym);
855 const char *get_param_name(struct sm_state *sm);
856 const char *get_mtag_name_var_sym(const char *state_name, struct symbol *sym);
857 const char *get_mtag_name_expr(struct expression *expr);
858 char *get_data_info_name(struct expression *expr);

860 char *escape_newlines(const char *str);
861 void sql_exec(struct sqlite3 *db, int (*callback)(void*, int, char**, char**), v

863 #define sql_helper(db, call_back, data, sql...)
864 do {
865         char sql_txt[1024];
866
867         sqlite3_snprintf(sizeof(sql_txt), sql_txt, sql);
868         sm_debug("debug: %s\n", sql_txt);
869         sql_exec(db, call_back, data, sql_txt);
870 } while (0)


873 #define run_sql(call_back, data, sql...)
874 do {
875         if (option_no_db)
876                 break;
877         sql_helper(smatch_db, call_back, data, sql);
878 } while (0)

880 #define mem_sql(call_back, data, sql...)
881         sql_helper(mem_db, call_back, data, sql)

883 #define cache_sql(call_back, data, sql...)
884         sql_helper(cache_db, call_back, data, sql)

886 #define sql_insert_helper(table, db, ignore, late, values...)
887 do {
888         struct sqlite3 *_db = db;
889
890         if (__inline_fn && !_db)
891                 _db = mem_db;
892         if (_db) {
893                 char buf[1024];
894                 char *err, *p = buf;
895                 int rc;
896
897                 p += snprintf(p, buf + sizeof(buf) - p,
898                               "insert %sinto %s values (",
899                               ignore ? "or ignore " : "", #table);
900                 p += snprintf(p, buf + sizeof(buf) - p, values);
901                 p += snprintf(p, buf + sizeof(buf) - p, ");");
902                 sm_debug("mem-db: %s\n", buf);
903                 rc = sqlite3_exec(_db, buf, NULL, NULL, &err);
904                 if (rc != SQLITE_OK) {
905                         sm_ierror("SQL error #2: %s", err);
906                         sm_ierror("SQL: '%s'", buf);
907                         parse_error = 1;
908                 }
909                 break;
910         }
911         if (option_info) {
912                 FILE *tmp_fd = sm_outfd;
913                 sm_outfd = sql_outfd;
914                 sm_prefix();
915                 sm_printf("SQL%s: insert %sinto " #table " values(",
916                           late ? "_late" : "", ignore ? "or ignore " : "");
917                 sm_printf(values);
918                 sm_printf(");\n");
```

```
                sm_outfd = tmp_fd;
        }
} while (0)

#define sql_insert(table, values...) sql_insert_helper(table, 0, 0, 0, values);
#define sql_insert_or_ignore(table, values...) sql_insert_helper(table, 0, 1, 0,
#define sql_insert_late(table, values...) sql_insert_helper(table, 0, 0, 1, valu
#define sql_insert_cache(table, values...) sql_insert_helper(table, cache_db, 1,

char *get_static_filter(struct symbol *sym);

void sql_insert_return_states(int return_id, const char *return_ranges,
                int type, int param, const char *key, const char *value);
void sql_insert_caller_info(struct expression *call, int type, int param,
                const char *key, const char *value);
void sql_insert_function_ptr(const char *fn, const char *struct_name);
void sql_insert_return_values(const char *return_values);
void sql_insert_return_implies(int type, int param, const char *key, const char
void sql_insert_function_type_size(const char *member, const char *ranges);
void sql_insert_function_type_info(int type, const char *struct_type, const char
void sql_insert_type_info(int type, const char *member, const char *value);
void sql_insert_local_values(const char *name, const char *value);
void sql_insert_function_type_value(const char *type, const char *value);
void sql_insert_function_type(int param, const char *value);
void sql_insert_parameter_name(int param, const char *value);
void sql_insert_data_info(struct expression *data, int type, const char *value);
void sql_insert_data_info_var_sym(const char *var, struct symbol *sym, int type,
void sql_save_constraint(const char *con);
void sql_save_constraint_required(const char *data, int op, const char *limit);
void sql_copy_constraint_required(const char *new_limit, const char *old_limit);
void sql_insert_fn_ptr_data_link(const char *ptr, const char *data);
void sql_insert_fn_data_link(struct expression *fn, int type, int param, const c
void sql_insert_mtag_about(mtag_t tag, const char *left_name, const char *right_
void insert_mtag_data(sval_t sval, struct range_list *rl);
void sql_insert_mtag_map(mtag_t tag, int offset, mtag_t container);
void sql_insert_mtag_alias(mtag_t orig, mtag_t alias);
int mtag_map_select_container(mtag_t tag, int offset, mtag_t *container);
int mtag_map_select_tag(mtag_t container, int offset, mtag_t *tag);

void sql_select_return_states(const char *cols, struct expression *call,
        int (*callback)(void*, int, char**, char**), void *info);
void sql_select_call_implies(const char *cols, struct expression *call,
        int (*callback)(void*, int, char**, char**));

void open_smatch_db(char *db_file);

/* smatch_files.c */
int open_data_file(const char *filename);
int open_schema_file(const char *schema);
struct token *get_tokens_file(const char *filename);

/* smatch.c */
extern char *option_debug_check;
extern char *option_project_str;
extern char *bin_dir;
extern char *data_dir;
extern int option_no_data;
extern int option_full_path;
extern int option_param_mapper;
extern int option_call_tree;
extern int num_checks;

enum project_type {
        PROJ_NONE,
        PROJ_KERNEL,
        PROJ_WINE,
```



```
        PROJ_ILLUMOS_KERNEL,
        PROJ_ILLUMOS_USER,
        PROJ_UNKNOWN,
};
extern enum project_type option_project;
const char *check_name(unsigned short id);
int id_from_name(const char *name);


/* smatch_buf_size.c */
int get_array_size(struct expression *expr);
int get_array_size_bytes(struct expression *expr);
int get_array_size_bytes_min(struct expression *expr);
int get_array_size_bytes_max(struct expression *expr);
struct range_list *get_array_size_bytes_rl(struct expression *expr);
int get_real_array_size(struct expression *expr);
int last_member_is_resizable(struct symbol *type);
/* smatch_strlen.c */
int get_implied_strlen(struct expression *expr, struct range_list **rl);
int get_size_from_strlen(struct expression *expr);

/* smatch_capped.c */
int is_capped(struct expression *expr);
int is_capped_var_sym(const char *name, struct symbol *sym);

/* check_user_data.c */
int is_user_macro(struct expression *expr);
int is_user_data(struct expression *expr);
int is_capped_user_data(struct expression *expr);
int implied_user_data(struct expression *expr, struct range_list **rl);
struct stree *get_user_stree(void);
int get_user_rl(struct expression *expr, struct range_list **rl);
int get_user_rl_spammy(struct expression *expr, struct range_list **rl);
int is_user_rl(struct expression *expr);
int get_user_rl_var_sym(const char *name, struct symbol *sym, struct range_list

/* check_locking.c */
void print_held_locks();

/* check_assigned_expr.c */
struct expression *get_assigned_expr(struct expression *expr);
struct expression *get_assigned_expr_name_sym(const char *name, struct symbol *s
/* smatch_return_to_param.c */
void __add_return_to_param_mapping(struct expression *assign, const char *return
char *map_call_to_param_name_sym(struct expression *expr, struct symbol **sym);

/* smatch_comparison.c */
struct compare_data {
        /* The ->left and ->right expression pointers might be NULL (I'm lazy) *
        struct expression *left;
        const char *left_var;
        struct var_sym_list *left_vsl;
        int comparison;
        struct expression *right;
        const char *right_var;
        struct var_sym_list *right_vsl;
};
DECLARE_ALLOCATOR(compare_data);
struct smatch_state *alloc_compare_state(
                struct expression *left,
                const char *left_var, struct var_sym_list *left_vsl,
                int comparison,
                struct expression *right,
                const char *right_var, struct var_sym_list *right_vsl);
int filter_comparison(int orig, int op);
int merge_comparisons(int one, int two);
```

```
1051 int combine_comparisons(int left_compare, int right_compare);
1052 int state_to_comparison(struct smatch_state *state);
1053 struct smatch_state *merge_compare_states(struct smatch_state *s1, struct smatch
1054 int get_comparison(struct expression *left, struct expression *right);
1055 int get_comparison_strings(const char *one, const char *two);
1056 int possible_comparison(struct expression *a, int comparison, struct expression
1057 struct state_list *get_all_comparisons(struct expression *expr);
1058 struct state_list *get_all_possible_equal_comparisons(struct expression *expr);
1059 void __add_return_comparison(struct expression *call, const char *range);
1060 void __add_comparison_info(struct expression *expr, struct expression *call, con
1061 char *get_printed_param_name(struct expression *call, const char *param_name, st
1062 char *name_sym_to_param_comparison(const char *name, struct symbol *sym);
1063 char *expr_equal_to_param(struct expression *expr, int ignore);
1064 char *expr_lte_to_param(struct expression *expr, int ignore);
1065 char *expr_param_comparison(struct expression *expr, int ignore);
1066 int flip_comparison(int op);
1067 int negate_comparison(int op);
1068 int remove_unsigned_from_comparison(int op);
1069 int param_compare_limit_is_impossible(struct expression *expr, int left_param, c
1070 void filter_by_comparison(struct range_list **rl, int comparison, struct range_l
1071 struct sm_state *comparison_implication_hook(struct expression *expr,
1072                         struct state_list **true_stack,
1073                         struct state_list **false_stack);
1074 void __compare_param_limit_hook(struct expression *left_expr, struct expression
1075                                 const char *state_name,
1076                                 struct smatch_state *true_state, struct smatch_s
1077 int impossibly_high_comparison(struct expression *expr);

1079 /* smatch_sval.c */
1080 sval_t *sval_alloc(sval_t sval);
1081 sval_t *sval_alloc_permanent(sval_t sval);
1082 sval_t sval_blank(struct expression *expr);
1083 sval_t sval_type_val(struct symbol *type, long long val);
1084 sval_t sval_from_val(struct expression *expr, long long val);
1085 int sval_is_ptr(sval_t sval);
1086 int sval_unsigned(sval_t sval);
1087 int sval_signed(sval_t sval);
1088 int sval_bits(sval_t sval);
1089 int sval_bits_used(sval_t sval);
1090 int sval_is_negative(sval_t sval);
1091 int sval_is_positive(sval_t sval);
1092 int sval_is_min(sval_t sval);
1093 int sval_is_max(sval_t sval);
1094 int sval_is_a_min(sval_t sval);
1095 int sval_is_a_max(sval_t sval);
1096 int sval_is_negative_min(sval_t sval);
1097 int sval_cmp_t(struct symbol *type, sval_t one, sval_t two);
1098 int sval_cmp_val(sval_t one, long long val);
1099 sval_t sval_min(sval_t one, sval_t two);
1100 sval_t sval_max(sval_t one, sval_t two);
1101 int sval_too_low(struct symbol *type, sval_t sval);
1102 int sval_too_high(struct symbol *type, sval_t sval);
1103 int sval_fits(struct symbol *type, sval_t sval);
1104 sval_t sval_cast(struct symbol *type, sval_t sval);
1105 sval_t sval_preop(sval_t sval, int op);
1106 sval_t sval_binop(sval_t left, int op, sval_t right);
1107 int sval_binop_overflows(sval_t left, int op, sval_t right);
1108 int sval_binop_overflows_no_sign(sval_t left, int op, sval_t right);
1109 unsigned long long fls_mask(unsigned long long uvalue);
1110 unsigned long long sval_fls_mask(sval_t sval);
1111 const char *sval_to_str(sval_t sval);
1112 const char *sval_to_numstr(sval_t sval);
1113 sval_t ll_to_sval(long long val);

1115 /* smatch_string_list.c */
1116 int list_has_string(struct string_list *str_list, const char *str);
```



```
1117 void insert_string(struct string_list **str_list, const char *str);
1118 struct string_list *clone_str_list(struct string_list *orig);
1119 struct string_list *combine_string_lists(struct string_list *one, struct string_

1121 /* smatch_start_states.c */
1122 struct stree *get_start_states(void);

1124 /* smatch_recurse.c */
1125 int has_symbol(struct expression *expr, struct symbol *sym);
1126 int has_variable(struct expression *expr, struct expression *var);
1127 int has_inc_dec(struct expression *expr);

1129 /* smatch_stored_conditions.c */
1130 struct smatch_state *get_stored_condition(struct expression *expr);
1131 struct expression_list *get_conditions(struct expression *expr);
1132 struct sm_state *stored_condition_implication_hook(struct expression *expr,
1133                         struct state_list **true_stack,
1134                         struct state_list **false_stack);

1136 /* check_string_len.c */
1137 int get_formatted_string_size(struct expression *call, int arg);

1139 /* smatch_param_set.c */
1140 int param_was_set(struct expression *expr);
1141 int param_was_set_var_sym(const char *name, struct symbol *sym);
1142 /* smatch_param_filter.c */
1143 int param_has_filter_data(struct sm_state *sm);

1145 /* smatch_links.c */
1146 void set_up_link_functions(int id, int linkid);
1147 struct smatch_state *merge_link_states(struct smatch_state *s1, struct smatch_st
1148 void store_link(int link_id, const char *name, struct symbol *sym, const char *l

1150 /* smatch_auto_copy.c */
1151 void set_auto_copy(int owner);

1153 /* check_buf_comparison */
1154 struct expression *get_size_variable(struct expression *buf);
1155 struct expression *get_array_variable(struct expression *size);

1157 /* smatch_untracked_param.c */
1158 void mark_untracked(struct expression *expr, int param, const char *key, const c
1159 void add_untracked_param_hook(void (func)(struct expression *call, int param));
1160 void mark_all_params_untracked(int return_id, char *return_ranges, struct expres

1162 /* smatch_strings.c */
1163 struct state_list *get_strings(struct expression *expr);
1164 struct expression *fake_string_from_mtag(mtag_t tag);

1166 /* smatch_estate.c */
1167 int estate_get_single_value(struct smatch_state *state, sval_t *sval);

1169 /* smatch_address.c */
1170 int get_address_rl(struct expression *expr, struct range_list **rl);
1171 int get_member_offset(struct symbol *type, const char *member_name);
1172 int get_member_offset_from_deref(struct expression *expr);

1174 /* for now this is in smatch_used_parameter.c */
1175 void __get_state_hook(int owner, const char *name, struct symbol *sym);

1177 /* smatch_buf_comparison.c */
1178 int db_var_is_array_limit(struct expression *array, const char *name, struct var

1180 struct stree *get_all_return_states(void);
1181 struct stree_stack *get_all_return_strees(void);
1182 int on_atomic_dec_path(void);
```

```
int was_inced(const char *name, struct symbol *sym);

/* smatch_constraints.c */
char *get_constraint_str(struct expression *expr);
struct constraint_list *get_constraints(struct expression *expr);
char *unmet_constraint(struct expression *data, struct expression *offset);
char *get_required_constraint(const char *data_str);

/* smatch_container_of.c */
int get_param_from_container_of(struct expression *expr);
int get_offset_from_container_of(struct expression *expr);

/* smatch_mtag.c */
int get_string_mtag(struct expression *expr, mtag_t *tag);
int get_toplevel_mtag(struct symbol *sym, mtag_t *tag);
int get_mtag(struct expression *expr, mtag_t *tag);
int get_mtag_offset(struct expression *expr, mtag_t *tag, int *offset);
int create_mtag_alias(mtag_t tag, struct expression *expr, mtag_t *new);
int expr_to_mtag_offset(struct expression *expr, mtag_t *tag, int *offset);
void update_mtag_data(struct expression *expr);
int get_mtag_sval(struct expression *expr, sval_t *sval);
int get_mtag_addr_sval(struct expression *expr, sval_t *sval);

/* Trinity fuzzer stuff */
const char *get_syscall_arg_type(struct symbol *sym);

/* smatch_mem_tracker.c */
extern int option_mem;
unsigned long get_max_memory(void);

/* check_is_nospec.c */
bool is_nospec(struct expression *expr);

/* smatch_nul_terminator.c */
bool is_nul_terminated(struct expression *expr);

static inline int type_bits(struct symbol *type)
{
        if (!type)
                return 0;
        if (type->type == SYM_PTR)  /* Sparse doesn't set this for &pointers */
                return bits_in_pointer;
        if (type->type == SYM_ARRAY)
                return bits_in_pointer;
        if (!type->examined)
                examine_symbol_type(type);
        return type->bit_size;
}

static inline bool type_is_ptr(struct symbol *type)
{
        return type && (type->type == SYM_PTR || type->type == SYM_ARRAY);
}

static inline int type_unsigned(struct symbol *base_type)
{
        if (!base_type)
                return 0;
        if (base_type->ctype.modifiers & MOD_UNSIGNED)
                return 1;
        return 0;
}

static inline int type_positive_bits(struct symbol *type)
{
        if (!type)
```



```
                return 0;
        if (type->type == SYM_ARRAY)
                return bits_in_pointer - 1;
        if (type_unsigned(type))
                return type_bits(type);
        return type_bits(type) - 1;
}

static inline int sval_positive_bits(sval_t sval)
{
        return type_positive_bits(sval.type);
}

/*
 * Returns -1 if one is smaller, 0 if they are the same and 1 if two is larger.
 */
static inline int sval_cmp(sval_t one, sval_t two)
{
        struct symbol *type;

        type = one.type;
        if (sval_positive_bits(two) > sval_positive_bits(one))
                type = two.type;
        if (type_bits(type) < 31)
                type = &int_ctype;

        one = sval_cast(type, one);
        two = sval_cast(type, two);

        if (type_unsigned(type)) {
                if (one.uvalue < two.uvalue)
                        return -1;
                if (one.uvalue == two.uvalue)
                        return 0;
                return 1;
        }
        /* fix me handle type promotion and unsigned values */
        if (one.value < two.value)
                return -1;
        if (one.value == two.value)
                return 0;
        return 1;
}

#endif      /* !SMATCH_H_ */
```

```
**********************************************************
    5345 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch_about_fn_ptr_arg.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Say you have assign a function to a function pointer and you assign a
 * pointer to the data argument then we want to record some information about
 * the argument.  Right now what I mainly want to record is the type of it, I
 * guess.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"
#include <ctype.h>

static int my_id;

static int assigns_parameters(struct expression *fn, struct expression *arg)
{
        int fn_param, arg_param;
        char buf[32];

        fn_param = get_param_num(fn);
        if (fn_param < 0)
                return 0;

        arg_param = get_param_num(arg);
        if (arg_param < 0)
                return 0;

        snprintf(buf, sizeof(buf), "%d", arg_param);
        sql_insert_return_implies(FN_ARG_LINK, fn_param, "$", buf);
        return 1;
}

static void link_function_arg(struct expression *fn, int param, struct expressio
{
        struct symbol *type;

        if (!fn || !arg)
                return;
        if (assigns_parameters(fn, arg))
                return;

        type = get_type(arg);
```



```
        if (!type || type->type != SYM_PTR)
                return;
        type = get_real_base_type(type);
        if (!type)
                return;
        // FIXME: param shouldn't always be 0?
        sql_insert_fn_data_link(fn, PASSES_TYPE, param, "$", type_to_str(type));
}

char *next_param_name;
struct symbol *next_param_sym;
struct expression *next_fn;
static void match_assign_param(struct expression *expr)
{
        struct symbol *sym;
        char *name;

        if (!next_param_name)
                return;

        name = expr_to_var_sym(expr->left, &sym);
        if (!name || !sym) {
                free_string(name);
                return;
        }

        if (sym != next_param_sym ||
            strcmp(name, next_param_name) != 0)
                return;

        link_function_arg(next_fn, 0, strip_expr(expr->right));

        next_param_name = 0;
        next_param_sym = NULL;
        next_fn = NULL;
}

static int get_arg_ptr(void *_arg_ptr, int argc, char **argv, char **azColName)
{
        char **arg_ptr = _arg_ptr;

        *arg_ptr = NULL;
        if (argc != 1)
                return 0;
        *arg_ptr = alloc_string(argv[0]);
        return 0;
}

static char *get_data_member(char *fn_member, struct expression *expr, struct sy
{
        struct symbol *tmp_sym;
        char *fn_str;
        char *arg_ptr = NULL;
        char *end_type;
        int len_ptr, len_str;
        char buf[128];

        *sym = NULL;
        run_sql(get_arg_ptr, &arg_ptr,
                "select data from fn_ptr_data_link where fn_ptr = '%s';", fn_mem
        if (!arg_ptr)
                return NULL;
        end_type = strchr(arg_ptr, '>');
        if (!end_type)
                return NULL;
        end_type++;
```

```
127         fn_str = expr_to_var_sym(expr, &tmp_sym);
128         if (!fn_str || !tmp_sym)
129                 return NULL;
130         len_ptr = strlen(fn_member);
131         len_str = strlen(fn_str);
132         while (len_str > 0 && len_ptr > 0) {
133                 if (fn_str[len_str - 1] != fn_member[len_ptr - 1])
134                         break;
135                 if (fn_str[len_str - 1] == '>')
136                         break;
137                 len_str--;
138                 len_ptr--;
139         }
141         strncpy(buf, fn_str, sizeof(buf));
142         snprintf(buf + len_str, sizeof(buf) - len_str, end_type);
143         *sym = tmp_sym;
144         return alloc_string(buf);
145 }
147 static void match_assign_function(struct expression *expr)
148 {
149         struct expression *right, *arg;
150         struct symbol *sym;
151         char *data_member;
152         struct symbol *type;
153         char *member_name;
155         right = strip_expr(expr->right);
156         if (right->type == EXPR_PREOP && right->op == '&')
157                 right = strip_expr(right->unop);
159         type = get_type(right);
160         if (type && type->type == SYM_PTR)
161                 type = get_real_base_type(type);
162         if (!type || type->type != SYM_FN)
163                 return;
165         member_name = get_member_name(expr->left);
166         if (!member_name)
167                 return;
169         data_member = get_data_member(member_name, expr->left, &sym);
170         if (!data_member || !sym) {
171                 free_string(data_member);
172                 data_member = NULL;
173         }
175         arg = get_assigned_expr_name_sym(data_member, sym);
176         if (arg) {
177                 link_function_arg(right, 0, arg);
178         } else {
179                 next_param_name = data_member;
180                 next_param_sym = sym;
181                 next_fn = right;
182         }
183 }
185 static int is_recursive_call(struct expression *call)
186 {
187         if (call->fn->type != EXPR_SYMBOL)
188                 return 0;
189         if (call->fn->symbol == cur_func_sym)
190                 return 1;
191         return 0;
192 }
```



```
194 static void check_passes_fn_and_data(struct expression *call, struct expression
195 {
196         struct expression *arg;
197         struct symbol *type;
198         int data_nr;
200         if (is_recursive_call(call))
201                 return;
203         type = get_type(fn);
204         if (!type || type->type != SYM_FN)
205                 return;
207         if (!isdigit(value[0]))
208                 return;
209         data_nr = atoi(value);
210         arg = get_argument_from_call_expr(call->args, data_nr);
211         if (!arg)
212                 return;
213         link_function_arg(fn, 0, arg);
214 }
216 static void match_end_func(struct symbol *sym)
217 {
218         next_param_sym = NULL;
219         next_fn = NULL;
220 }
222 void register_about_fn_ptr_arg(int id)
223 {
224         my_id = id;
226         if (0 && !option_info)
227                 return;
228         add_hook(match_assign_param, ASSIGNMENT_HOOK);
229         add_hook(match_assign_function, ASSIGNMENT_HOOK);
230         select_return_implies_hook(FN_ARG_LINK, &check_passes_fn_and_data);
231         add_hook(&match_end_func, END_FUNC_HOOK);
232 }
```

```
**********************************************************
    5999 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch_address.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static bool is_non_null_array(struct expression *expr)
{
        struct symbol *type;
        struct symbol *sym;
        struct symbol *tmp;
        int i;

        type = get_type(expr);
        if (!type || type->type != SYM_ARRAY)
                return 0;
        if (expr->type == EXPR_SYMBOL)
                return 1;
        if (implied_not_equal(expr, 0))
                return 1;

        /* verify that it's not the first member of the struct */
        if (expr->type != EXPR_DEREF || !expr->member)
                return 0;
        sym = expr_to_sym(expr);
        if (!sym)
                return 0;
        type = get_real_base_type(sym);
        if (!type || type->type != SYM_PTR)
                return 0;
        type = get_real_base_type(type);
        if (type->type != SYM_STRUCT)
                return 0;

        i = 0;
        FOR_EACH_PTR(type->symbol_list, tmp) {
                i++;
                if (!tmp->ident)
                        continue;
                if (strcmp(expr->member->name, tmp->ident->name) == 0) {
                        if (i == 1)
                                return 0;
                        return 1;
                }
        } END_FOR_EACH_PTR(tmp);
```



```
        return 0;
}

int get_member_offset(struct symbol *type, const char *member_name)
{
        struct symbol *tmp;
        int offset;

        if (!type || type->type != SYM_STRUCT)
                return -1;

        offset = 0;
        FOR_EACH_PTR(type->symbol_list, tmp) {
                offset = ALIGN(offset, tmp->ctype.alignment);
                if (tmp->ident &&
                    strcmp(member_name, tmp->ident->name) == 0) {
                        return offset;
                }
                offset += type_bytes(tmp);
        } END_FOR_EACH_PTR(tmp);
        return -1;
}

int get_member_offset_from_deref(struct expression *expr)
{
        struct symbol *type;
        struct ident *member;
        int offset;

        if (expr->type != EXPR_DEREF)  /* hopefully, this doesn't happen */
                return -1;

        if (expr->member_offset >= 0)
                return expr->member_offset;

        member = expr->member;
        if (!member)
                return -1;

        type = get_type(expr->deref);
        if (!type || type->type != SYM_STRUCT)
                return -1;

        offset = get_member_offset(type, member->name);
        if (offset >= 0)
                expr->member_offset = offset;
        return offset;
}

static struct range_list *filter_unknown_negatives(struct range_list *rl)
{
        struct data_range *first;
        struct range_list *filter = NULL;

        first = first_ptr_list((struct ptr_list *)rl);

        if (sval_is_min(first->min) &&
            sval_is_negative(first->max) &&
            first->max.value == -1) {
                add_ptr_list(&filter, first);
                return rl_filter(rl, filter);
        }

        return rl;
}
```

```
128 static void add_offset_to_pointer(struct range_list **rl, int offset)
129 {
130         sval_t min, max, remove, sval;
131         struct range_list *orig = *rl;
132
133         /*
134          * Ha ha.  Treating zero as a special case means I'm correct at least a
135          * tiny fraction of the time.  Which is better than nothing.
136          *
137          */
138         if (offset == 0)
139                 return;
140
141         /*
142          * This function doesn't necessarily work how you might expect...
143          *
144          * Say you have s64min-(-1),1-s64max and you add 8 then I guess what
145          * we want to say is maybe something like 9-s64max.  This shows that the
146          * min it could be is 9 which is potentially useful information.  But
147          * if we start with (-12),5000000-57777777 and we add 8 then we'd want
148          * the result to be (-4),5000008-57777777 but (-4),5000000-57777777 is
149          * also probably acceptable.  If you start with s64min-s64max then the
150          * result should be 8-s64max.
151          *
152          */
153
154         /* We do the math on void pointer type, because this isn't "&v + 16" it
155          * is &v->sixteenth_byte.
156          */
157         orig = cast_rl(&ptr_ctype, orig);
158         min = sval_type_min(&ptr_ctype);
159         min.value = offset;
160         max = sval_type_max(&ptr_ctype);
161
162         if (!orig || is_whole_rl(orig)) {
163                 *rl = alloc_rl(min, max);
164                 return;
165         }
166
167         orig = filter_unknown_negatives(orig);
168         /*
169          * FIXME:  This is not really accurate but we're a bit screwed anyway
170          * when we start doing pointer math with error pointers so it's probably
171          * not important.
172          *
173          */
174         if (sval_is_negative(rl_min(orig)))
175                 return;
176
177         /* no wrap around */
178         max.uvalue = rl_max(orig).uvalue;
179         if (max.uvalue > sval_type_max(&ptr_ctype).uvalue - offset) {
180                 remove = sval_type_max(&ptr_ctype);
181                 remove.uvalue -= offset;
182                 orig = remove_range(orig, remove, max);
183         }
184
185         sval.type = &int_ctype;
186         sval.value = offset;
187
188         *rl = rl_binop(orig, '+', alloc_rl(sval, sval));
189 }
190
191 static struct range_list *where_allocated_rl(struct symbol *sym)
192 {
```



```
193         if (!sym)
194                 return NULL;
195
196         if (sym->ctype.modifiers & (MOD_TOPLEVEL | MOD_STATIC)) {
197                 if (sym->initializer)
198                         return alloc_rl(data_seg_min, data_seg_max);
199                 else
200                         return alloc_rl(bss_seg_min, bss_seg_max);
201         }
202         return alloc_rl(stack_seg_min, stack_seg_max);
203 }
204
205 int get_address_rl(struct expression *expr, struct range_list **rl)
206 {
207         expr = strip_expr(expr);
208         if (!expr)
209                 return 0;
210
211         if (expr->type == EXPR_STRING) {
212                 *rl = alloc_rl(text_seg_min, text_seg_max);
213                 return 1;
214         }
215
216         if (expr->type == EXPR_PREOP && expr->op == '&') {
217                 struct expression *unop;
218
219                 unop = strip_expr(expr->unop);
220                 if (unop->type == EXPR_SYMBOL) {
221                         *rl = where_allocated_rl(unop->symbol);
222                         return 1;
223                 }
224
225                 if (unop->type == EXPR_DEREF) {
226                         int offset = get_member_offset_from_deref(unop);
227
228                         unop = strip_expr(unop->unop);
229                         if (unop->type == EXPR_SYMBOL) {
230                                 *rl = where_allocated_rl(unop->symbol);
231                         } else if (unop->type == EXPR_PREOP && unop->op == '*')
232                                 unop = strip_expr(unop->unop);
233                                 get_absolute_rl(unop, rl);
234                         } else {
235                                 return 0;
236                         }
237
238                         add_offset_to_pointer(rl, offset);
239                         return 1;
240                 }
241
242                 return 0;
243         }
244
245         if (is_non_null_array(expr)) {
246                 *rl = alloc_rl(array_min_sval, array_max_sval);
247                 return 1;
248         }
249
250         return 0;
251 }
```

**********
1491 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch_annotate.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * A place to add function annotations for common functions.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"

static int param_caps_return(struct expression *call, void *_arg, struct range_l
{
        int arg = PTR_INT(_arg);
        struct expression *expr;
        struct range_list *rl;

        expr = get_argument_from_call_expr(call->args, arg);
        if (get_implied_rl(expr, &rl) && rl_max(rl).value != 0)
                *res = alloc_rl(sval_type_val(rl_type(rl), 0), rl_max(rl));
        return 1;
}

void register_annotate(int id)
{
        /*
         * Technically snprintf() returns the number of bytes which *would* have
         * been printed.  I do try caclulating that in check_snprintf().  But
         * it probably works better to assume the limitter is accurate.
         */
        add_implied_return_hook("snprintf", &param_caps_return, INT_PTR(1));

}
```

```
**********************************************************
    4499 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch_array_values.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2018 Oracle.  All rights reserved.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

struct db_info {
        int count;
        struct symbol *type;
        struct range_list *rl;
};

static int get_vals(void *_db_info, int argc, char **argv, char **azColName)
{
        struct db_info *db_info = _db_info;
        struct range_list *rl;

        str_to_rl(db_info->type, argv[0], &rl);
        db_info->rl = rl_union(db_info->rl, rl);

        return 0;
}

static int is_file_local(struct expression *array)
{
        struct symbol *sym = NULL;
        char *name;

        name = expr_to_str_sym(array, &sym);
        free_string(name);
        if (!sym)
                return 0;

        if ((sym->ctype.modifiers & MOD_TOPLEVEL) &&
            (sym->ctype.modifiers & MOD_STATIC))
                return 1;
        return 0;
}

static char *get_toplevel_name(struct expression *array)
{
        char *name;
        char buf[128];
```



```
        if (is_array(array))
                array = get_array_base(array);

        if (!array || array->type != EXPR_SYMBOL)
                return NULL;
        if (!is_file_local(array))
                return NULL;

        name = expr_to_str(array);
        snprintf(buf, sizeof(buf), "%s[]", name);
        free_string(name);

        return alloc_sname(buf);
}

static char *get_member_array(struct expression *array)
{
        char *name;
        char buf[128];

        name = get_member_name(array);
        if (!name)
                return NULL;
        snprintf(buf, sizeof(buf), "%s[]", name);
        free_string(name);
        return alloc_sname(buf);
}

static char *get_array_name(struct expression *array)
{
        struct symbol *type;
        char *name;

        type = get_type(array);
        if (!type || type->type != SYM_ARRAY)
                return NULL;

        name = get_toplevel_name(array);
        if (name)
                return name;
        name = get_member_array(array);
        if (name)
                return name;

        return NULL;
}

int get_array_rl(struct expression *expr, struct range_list **rl)
{
        struct expression *array;
        struct symbol *type;
        struct db_info db_info = {};
        char *name;

        type = get_type(expr);
        if (!type || type->type != SYM_BASETYPE)
                return 0;
        db_info.type = type;

        array = get_array_base(expr);
        name = get_array_name(array);
        if (!name)
                return 0;

        if (is_file_local(array)) {
```

```
127                 run_sql(&get_vals, &db_info,
128                         "select value from sink_info where file = '%s' and stati
129                         get_filename(), name, DATA_VALUE);
130         } else {
131                 run_sql(&get_vals, &db_info,
132                         "select value from sink_info where sink_name = '%s' and
133                         name, DATA_VALUE);
134         }
135         if (!db_info.rl || db_info.count >= 10)
136                 return 0;
137
138         *rl = db_info.rl;
139         return 1;
140 }
141
142 static struct range_list *get_saved_rl(struct symbol *type, char *name)
143 {
144         struct db_info db_info = {.type = type};
145
146         cache_sql(&get_vals, &db_info, "select value from sink_info where sink_n
147                   name, DATA_VALUE);
148         return db_info.rl;
149 }
150
151 static void update_cache(char *name, int is_static, struct range_list *rl)
152 {
153         cache_sql(NULL, NULL, "delete from sink_info where sink_name = '%s' and
154                   name, DATA_VALUE);
155         cache_sql(NULL, NULL, "insert into sink_info values ('%s', %d, '%s', %d,
156                   get_filename(), is_static, name, DATA_VALUE, show_rl(rl));
157 }
158
159 static void match_assign(struct expression *expr)
160 {
161         struct expression *left, *array;
162         struct range_list *orig_rl, *rl;
163         struct symbol *type;
164         char *name;
165
166         type = get_type(expr->right);
167         if (!type || type->type != SYM_BASETYPE)
168                 return;
169
170         left = strip_expr(expr->left);
171         if (!is_array(left))
172                 return;
173         array = get_array_base(left);
174         name = get_array_name(array);
175         if (!name)
176                 return;
177
178         if (expr->op != '=') {
179                 rl = alloc_whole_rl(type);
180         } else {
181                 get_absolute_rl(expr->right, &rl);
182                 rl = cast_rl(type, rl);
183                 orig_rl = get_saved_rl(type, name);
184                 rl = rl_union(orig_rl, rl);
185         }
186
187         update_cache(name, is_file_local(array), rl);
188 }
189
190 void register_array_values(int id)
191 {
192         my_id = id;
```



```
193
194         add_hook(&match_assign, ASSIGNMENT_HOOK);
195         add_hook(&match_assign, GLOBAL_ASSIGNMENT_HOOK);
196 }
```

```
**********************************************************
    3439 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch_assigned_expr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is not a check.  It just saves an struct expression pointer
 * whenever something is assigned.  This can be used later on by other scripts.
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

int check_assigned_expr_id;
static int my_id;
static int link_id;

static void undef(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &undefined);
}

struct expression *get_assigned_expr(struct expression *expr)
{
        struct smatch_state *state;

        state = get_state_expr(my_id, expr);
        if (!state)
                return NULL;
        return (struct expression *)state->data;
}

struct expression *get_assigned_expr_name_sym(const char *name, struct symbol *s
{
        struct smatch_state *state;

        state = get_state(my_id, name, sym);
        if (!state)
                return NULL;
        return (struct expression *)state->data;
}

static void match_assignment(struct expression *expr)
{
        struct symbol *left_sym, *right_sym;
        char *left_name = NULL;
        char *right_name = NULL;
```



```
        if (expr->op != '=')
                return;
        if (is_fake_call(expr->right))
                return;
        if (__in_fake_struct_assign) {
                struct range_list *rl;

                if (!get_implied_rl(expr->right, &rl))
                        return;
                if (is_whole_rl(rl))
                        return;
        }

        left_name = expr_to_var_sym(expr->left, &left_sym);
        if (!left_name || !left_sym)
                goto free;
        set_state(my_id, left_name, left_sym, alloc_state_expr(strip_expr(expr->

        right_name = expr_to_var_sym(expr->right, &right_sym);
        if (!right_name || !right_sym)
                goto free;

        store_link(link_id, right_name, right_sym, left_name, left_sym);

free:
        free_string(left_name);
        free_string(right_name);
}

static void record_param_assignment(struct expression *expr, int param, char *ke
{
        struct expression *arg, *right;
        struct symbol *sym;
        char *name;
        char *p;
        int right_param;

        while (expr->type == EXPR_ASSIGNMENT)
                expr = strip_expr(expr->right);
        if (!expr || expr->type != EXPR_CALL)
                return;

        p = strstr(value, "[$");
        if (!p)
                return;

        p += 2;
        right_param = strtol(p, &p, 10);
        if (*p != ']')
                return;

        arg = get_argument_from_call_expr(expr->args, param);
        right = get_argument_from_call_expr(expr->args, right_param);
        if (!right || !arg)
                return;
        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        set_state(my_id, name, sym, alloc_state_expr(right));
free:
        free_string(name);
}

void register_assigned_expr(int id)
```

```
127 {
128         my_id = check_assigned_expr_id = id;
129         add_hook(&match_assignment, ASSIGNMENT_HOOK_AFTER);
130         add_modification_hook(my_id, &undef);
131         select_return_states_hook(PARAM_SET, &record_param_assignment);
132 }

134 void register_assigned_expr_links(int id)
135 {
136         link_id = id;
137         db_ignore_states(link_id);
138         set_up_link_functions(my_id, link_id);
139 }
```

```
**********************************************************
    2021 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch_auto_copy.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static int *auto_copy;

void set_auto_copy(int owner)
{
        if (owner <= 1 || owner > num_checks) {
                sm_ierror("bogus set_auto_copy()");
                return;
        }
        auto_copy[owner] = 1;
}

static void match_assign(struct expression *expr)
{
        char *left_name = NULL;
        char *right_name = NULL;
        struct symbol *left_sym, *right_sym;
        struct state_list *slist = NULL;
        struct sm_state *sm;

        left_name = expr_to_var_sym(expr->left, &left_sym);
        if (!left_name || !left_sym)
                goto free;
        right_name = expr_to_var_sym(expr->right, &right_sym);
        if (!right_name || !right_sym)
                goto free;

        FOR_EACH_SM(__get_cur_stree(), sm) {
                if (sm->owner <= 1 || sm->owner > num_checks)
                        continue;
                if (!auto_copy[sm->owner])
                        continue;
                if (right_sym != sm->sym)
                        continue;
                if (strcmp(right_name, sm->name) != 0)
                        continue;
                add_ptr_list(&slist, sm);
        } END_FOR_EACH_SM(sm);
```



```
        FOR_EACH_PTR(slist, sm) {
                set_state(sm->owner, left_name, left_sym, sm->state);
        } END_FOR_EACH_PTR(sm);

free:
        free_slist(&slist);
        free_string(left_name);
        free_string(right_name);
}

void register_auto_copy(int id)
{
        my_id = id;
        auto_copy = malloc((num_checks + 1) * sizeof(*auto_copy));
        memset(auto_copy, 0, (num_checks + 1) * sizeof(*auto_copy));

        add_hook(&match_assign, ASSIGNMENT_HOOK);
}
```

```
**********************************************************
   15038 Fri Dec 21 15:00:15 2018
new/usr/src/tools/smatch/src/smatch_buf_comparison.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The point here is to store that a buffer has x bytes even if we don't know
 * the value of x.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int size_id;
static int link_id;

/*
 * We need this for code which does:
 *
 *     if (size)
 *         foo = malloc(size);
 *
 * We want to record that the size of "foo" is "size" even after the merge.
 *
 */
static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        struct expression *size_expr;
        sval_t sval;

        if (!sm->state->data)
                return &undefined;
        size_expr = sm->state->data;
        if (!get_implied_value(size_expr, &sval) || sval.value != 0)
                return &undefined;
        return sm->state;
}

static struct smatch_state *merge_links(struct smatch_state *s1, struct smatch_s
{
        struct expression *expr1, *expr2;

        expr1 = s1->data;
        expr2 = s2->data;

        if (expr1 && expr2 && expr_equiv(expr1, expr2))
```



```
                return s1;
        return &merged;
}

static void match_link_modify(struct sm_state *sm, struct expression *mod_expr)
{
        struct expression *expr;
        struct sm_state *tmp;

        expr = sm->state->data;
        if (expr) {
                set_state_expr(size_id, expr, &undefined);
                set_state(link_id, sm->name, sm->sym, &undefined);
                return;
        }

        FOR_EACH_PTR(sm->possible, tmp) {
                expr = tmp->state->data;
                if (expr)
                        set_state_expr(size_id, expr, &undefined);
        } END_FOR_EACH_PTR(tmp);
        set_state(link_id, sm->name, sm->sym, &undefined);
}

static struct smatch_state *alloc_expr_state(struct expression *expr)
{
        struct smatch_state *state;
        char *name;

        state = __alloc_smatch_state(0);
        expr = strip_expr(expr);
        name = expr_to_str(expr);
        state->name = alloc_sname(name);
        free_string(name);
        state->data = expr;
        return state;
}

static int bytes_per_element(struct expression *expr)
{
        struct symbol *type;

        type = get_type(expr);
        if (!type)
                return 0;

        if (type->type != SYM_PTR && type->type != SYM_ARRAY)
                return 0;

        type = get_base_type(type);
        return type_bytes(type);
}

static void db_save_type_links(struct expression *array, struct expression *size
{
        const char *array_name;

        array_name = get_data_info_name(array);
        if (!array_name)
                array_name = "";
        sql_insert_data_info(size, ARRAY_LEN, array_name);
}

static void match_alloc_helper(struct expression *pointer, struct expression *si
{
        struct expression *tmp;
```

```
127         struct sm_state *sm;
128         sval_t sval;
129         int cnt = 0;

131         pointer = strip_expr(pointer);
132         size = strip_expr(size);
133         if (!size || !pointer)
134                 return;

136         while ((tmp = get_assigned_expr(size))) {
137                 size = strip_expr(tmp);
138                 if (cnt++ > 5)
139                         break;
140         }

142         if (size->type == EXPR_BINOP && size->op == '*') {
143                 struct expression *mult_left, *mult_right;

145                 mult_left = strip_expr(size->left);
146                 mult_right = strip_expr(size->right);

148                 if (get_implied_value(mult_left, &sval) &&
149                     sval.value == bytes_per_element(pointer))
150                         size = mult_right;
151                 else if (get_implied_value(mult_right, &sval) &&
152                     sval.value == bytes_per_element(pointer))
153                         size = mult_left;
154                 else
155                         return;
156         }

158         /* Only save links to variables, not fixed sizes */
159         if (get_value(size, &sval))
160                 return;

162         db_save_type_links(pointer, size);
163         sm = set_state_expr(size_id, pointer, alloc_expr_state(size));
164         if (!sm)
165                 return;
166         set_state_expr(link_id, size, alloc_expr_state(pointer));
167 }

169 static void match_alloc(const char *fn, struct expression *expr, void *_size_arg
170 {
171         int size_arg = PTR_INT(_size_arg);
172         struct expression *pointer, *call, *arg;

174         pointer = strip_expr(expr->left);
175         call = strip_expr(expr->right);
176         arg = get_argument_from_call_expr(call->args, size_arg);
177         match_alloc_helper(pointer, arg);
178 }

180 static void match_calloc(const char *fn, struct expression *expr, void *_start_a
181 {
182         int start_arg = PTR_INT(_start_arg);
183         struct expression *pointer, *call, *arg;
184         struct sm_state *tmp;
185         sval_t sval;

187         pointer = strip_expr(expr->left);
188         call = strip_expr(expr->right);
189         arg = get_argument_from_call_expr(call->args, start_arg);
190         if (get_implied_value(arg, &sval) &&
191             sval.value == bytes_per_element(pointer))
192                 arg = get_argument_from_call_expr(call->args, start_arg + 1);
```



```
194         db_save_type_links(pointer, arg);
195         tmp = set_state_expr(size_id, pointer, alloc_expr_state(arg));
196         if (!tmp)
197                 return;
198         set_state_expr(link_id, arg, alloc_expr_state(pointer));
199 }

201 struct expression *get_size_variable(struct expression *buf)
202 {
203         struct smatch_state *state;

205         state = get_state_expr(size_id, buf);
206         if (state)
207                 return state->data;
208         return NULL;
209 }

211 struct expression *get_array_variable(struct expression *size)
212 {
213         struct smatch_state *state;

215         state = get_state_expr(link_id, size);
216         if (state)
217                 return state->data;
218         return NULL;
219 }

221 static void array_check(struct expression *expr)
222 {
223         struct expression *array;
224         struct expression *size;
225         struct expression *offset;
226         char *array_str, *offset_str;

228         expr = strip_expr(expr);
229         if (!is_array(expr))
230                 return;

232         array = get_array_base(expr);
233         size = get_size_variable(array);
234         if (!size)
235                 return;
236         offset = get_array_offset(expr);
237         if (!possible_comparison(size, SPECIAL_EQUAL, offset))
238                 return;

240         array_str = expr_to_str(array);
241         offset_str = expr_to_str(offset);
242         sm_warning("potentially one past the end of array '%s[%s]'", array_str,
243         free_string(array_str);
244         free_string(offset_str);
245 }

247 struct db_info {
248         char *name;
249         int ret;
250 };

252 static int db_limitter_callback(void *_info, int argc, char **argv, char **azCol
253 {
254         struct db_info *info = _info;

256         /*
257          * If possible the limitters are tied to the struct they limit.  If we
258          * aren't sure which struct they limit then we use them as limitters for
```

```
259          * everything.
260          */
261         if (!info->name || argv[0][0] == '\0' || strcmp(info->name, argv[0]) ==
262                 info->ret = 1;
263         return 0;
264 }

266 static char *vsl_to_data_info_name(const char *name, struct var_sym_list *vsl)
267 {
268         struct var_sym *vs;
269         struct symbol *type;
270         static char buf[80];
271         const char *p;

273         if (ptr_list_size((struct ptr_list *)vsl) != 1)
274                 return NULL;
275         vs = first_ptr_list((struct ptr_list *)vsl);

277         type = get_real_base_type(vs->sym);
278         if (!type || type->type != SYM_PTR)
279                 goto top_level_name;
280         type = get_real_base_type(type);
281         if (!type || type->type != SYM_STRUCT)
282                 goto top_level_name;
283         if (!type->ident)
284                 goto top_level_name;

286         p = name;
287         while ((name = strstr(p, "->")))
288                 p = name + 2;

290         snprintf(buf, sizeof(buf),"(struct %s)->%s", type->ident->name, p);
291         return alloc_sname(buf);

293 top_level_name:
294         if (!(vs->sym->ctype.modifiers & MOD_TOPLEVEL))
295                 return NULL;
296         if (vs->sym->ctype.modifiers & MOD_STATIC)
297                 snprintf(buf, sizeof(buf),"static %s", name);
298         else
299                 snprintf(buf, sizeof(buf),"global %s", name);
300         return alloc_sname(buf);
301 }

303 int db_var_is_array_limit(struct expression *array, const char *name, struct var
304 {
305         char *size_name;
306         char *array_name = get_data_info_name(array);
307         struct db_info db_info = {.name = array_name,};

309         size_name = vsl_to_data_info_name(name, vsl);
310         if (!size_name)
311                 return 0;

313         run_sql(db_limitter_callback, &db_info,
314                 "select value from data_info where type = %d and data = '%s';",
315                 ARRAY_LEN, size_name);

317         return db_info.ret;
318 }

320 static int known_access_ok_comparison(struct expression *expr)
321 {
322         struct expression *array;
323         struct expression *size;
324         struct expression *offset;
```



```
325         int comparison;

327         array = get_array_base(expr);
328         size = get_size_variable(array);
329         if (!size)
330                 return 0;
331         offset = get_array_offset(expr);
332         comparison = get_comparison(size, offset);
333         if (comparison == '>' || comparison == SPECIAL_UNSIGNED_GT)
334                 return 1;

336         return 0;
337 }

339 static int known_access_ok_numbers(struct expression *expr)
340 {
341         struct expression *array;
342         struct expression *offset;
343         sval_t max;
344         int size;

346         array = get_array_base(expr);
347         offset = get_array_offset(expr);

349         size = get_array_size(array);
350         if (size <= 0)
351                 return 0;

353         get_absolute_max(offset, &max);
354         if (max.uvalue < size)
355                 return 1;
356         return 0;
357 }

359 static void array_check_data_info(struct expression *expr)
360 {
361         struct expression *array;
362         struct expression *offset;
363         struct state_list *slist;
364         struct sm_state *sm;
365         struct compare_data *comp;
366         char *offset_name;
367         const char *equal_name = NULL;

369         expr = strip_expr(expr);
370         if (!is_array(expr))
371                 return;

373         if (known_access_ok_numbers(expr))
374                 return;
375         if (known_access_ok_comparison(expr))
376                 return;

378         array = get_array_base(expr);
379         offset = get_array_offset(expr);
380         offset_name = expr_to_var(offset);
381         if (!offset_name)
382                 return;
383         slist = get_all_possible_equal_comparisons(offset);
384         if (!slist)
385                 goto free;

387         FOR_EACH_PTR(slist, sm) {
388                 comp = sm->state->data;
389                 if (strcmp(comp->left_var, offset_name) == 0) {
390                         if (db_var_is_array_limit(array, comp->right_var, comp->
```

```
391                                 equal_name = comp->right_var;
392                                 break;
393                         }
394                 } else if (strcmp(comp->right_var, offset_name) == 0) {
395                         if (db_var_is_array_limit(array, comp->left_var, comp->l
396                                 equal_name = comp->left_var;
397                                 break;
398                         }
399                 }
400         } END_FOR_EACH_PTR(sm);

402         if (equal_name) {
403                 char *array_name = expr_to_str(array);

405                 sm_warning("potential off by one '%s[]' limit '%s'", array_name,
406                 free_string(array_name);
407         }

409 free:
410         free_slist(&slist);
411         free_string(offset_name);
412 }

414 static void add_allocation_function(const char *func, void *call_back, int param
415 {
416         add_function_assign_hook(func, call_back, INT_PTR(param));
417 }

419 static char *buf_size_param_comparison(struct expression *array, struct expressi
420 {
421         struct expression *arg;
422         struct expression *size;
423         static char buf[32];
424         int i;

426         size = get_size_variable(array);
427         if (!size)
428                 return NULL;

430         i = -1;
431         FOR_EACH_PTR(args, arg) {
432                 i++;
433                 if (arg == array)
434                         continue;
435                 if (!expr_equiv(arg, size))
436                         continue;
437                 snprintf(buf, sizeof(buf), "==$%d", i);
438                 return buf;
439         } END_FOR_EACH_PTR(arg);

441         return NULL;
442 }

444 static void match_call(struct expression *call)
445 {
446         struct expression *arg;
447         char *compare;
448         int param;

450         param = -1;
451         FOR_EACH_PTR(call->args, arg) {
452                 param++;
453                 if (!is_pointer(arg))
454                         continue;
455                 compare = buf_size_param_comparison(arg, call->args);
456                 if (!compare)
```



```
457                         continue;
458                 sql_insert_caller_info(call, ARRAY_LEN, param, "$", compare);
459         } END_FOR_EACH_PTR(arg);
460 }

462 static int get_param(int param, char **name, struct symbol **sym)
463 {
464         struct symbol *arg;
465         int i;

467         i = 0;
468         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
469                 /*
470                  * this is a temporary hack to work around a bug (I think in spa
471                  * 2.6.37-rc1:fs/reiserfs/journal.o
472                  * If there is a function definition without parameter name foun
473                  * after a function implementation then it causes a crash.
474                  * int foo() {}
475                  * int bar(char *);
476                  */
477                 if (arg->ident->name < (char *)100)
478                         continue;
479                 if (i == param) {
480                         *name = arg->ident->name;
481                         *sym = arg;
482                         return TRUE;
483                 }
484                 i++;
485         } END_FOR_EACH_PTR(arg);

487         return FALSE;
488 }

490 static void set_param_compare(const char *array_name, struct symbol *array_sym,
491 {
492         struct expression *array_expr;
493         struct expression *size_expr;
494         struct symbol *size_sym;
495         char *size_name;
496         long param;
497         struct sm_state *tmp;

499         if (strncmp(value, "==$", 3) != 0)
500                 return;
501         param = strtol(value + 3, NULL, 10);
502         if (!get_param(param, &size_name, &size_sym))
503                 return;
504         array_expr = symbol_expression(array_sym);
505         size_expr = symbol_expression(size_sym);

507         tmp = set_state_expr(size_id, array_expr, alloc_expr_state(size_expr));
508         if (!tmp)
509                 return;
510         set_state_expr(link_id, size_expr, alloc_expr_state(array_expr));
511 }

513 static void set_arraysize_arg(const char *array_name, struct symbol *array_sym,
514 {
515         struct expression *array_expr;
516         struct expression *size_expr;
517         struct symbol *size_sym;
518         char *size_name;
519         long param;
520         struct sm_state *tmp;

522         param = strtol(key, NULL, 10);
```

```
523         if (!get_param(param, &size_name, &size_sym))
524                 return;
525         array_expr = symbol_expression(array_sym);
526         size_expr = symbol_expression(size_sym);
527
528         tmp = set_state_expr(size_id, array_expr, alloc_expr_state(size_expr));
529         if (!tmp)
530                 return;
531         set_state_expr(link_id, size_expr, alloc_expr_state(array_expr));
532 }
533
534 static void munge_start_states(struct statement *stmt)
535 {
536         struct state_list *slist = NULL;
537         struct sm_state *sm;
538         struct sm_state *poss;
539
540         FOR_EACH_MY_SM(size_id, __get_cur_stree(), sm) {
541                 if (sm->state != &merged)
542                         continue;
543                 /*
544                  * screw it.  let's just assume that if one caller passes the
545                  * size then they all do.
546                  */
547                 FOR_EACH_PTR(sm->possible, poss) {
548                         if (poss->state != &merged &&
549                             poss->state != &undefined) {
550                                 add_ptr_list(&slist, poss);
551                                 break;
552                         }
553                 } END_FOR_EACH_PTR(poss);
554         } END_FOR_EACH_SM(sm);
555
556         FOR_EACH_PTR(slist, sm) {
557                 set_state(size_id, sm->name, sm->sym, sm->state);
558         } END_FOR_EACH_PTR(sm);
559
560         free_slist(&slist);
561 }
562
563 void register_buf_comparison(int id)
564 {
565         size_id = id;
566
567         add_unmatched_state_hook(size_id, &unmatched_state);
568
569         add_allocation_function("malloc", &match_alloc, 0);
570         add_allocation_function("memdup", &match_alloc, 1);
571         add_allocation_function("realloc", &match_alloc, 1);
572         if (option_project == PROJ_KERNEL) {
573                 add_allocation_function("kmalloc", &match_alloc, 0);
574                 add_allocation_function("kzalloc", &match_alloc, 0);
575                 add_allocation_function("vmalloc", &match_alloc, 0);
576                 add_allocation_function("__vmalloc", &match_alloc, 0);
577                 add_allocation_function("sock_kmalloc", &match_alloc, 1);
578                 add_allocation_function("kmemdup", &match_alloc, 1);
579                 add_allocation_function("kmemdup_user", &match_alloc, 1);
580                 add_allocation_function("dma_alloc_attrs", &match_alloc, 1);
581                 add_allocation_function("pci_alloc_consistent", &match_alloc, 1)
582                 add_allocation_function("pci_alloc_coherent", &match_alloc, 1);
583                 add_allocation_function("devm_kmalloc", &match_alloc, 1);
584                 add_allocation_function("devm_kzalloc", &match_alloc, 1);
585                 add_allocation_function("kcalloc", &match_calloc, 0);
586                 add_allocation_function("devm_kcalloc", &match_calloc, 1);
587                 add_allocation_function("kmalloc_array", &match_calloc, 0);
588                 add_allocation_function("krealloc", &match_alloc, 1);
```



```
589         }
590
591         add_hook(&array_check, OP_HOOK);
592         add_hook(&array_check_data_info, OP_HOOK);
593
594         add_hook(&match_call, FUNCTION_CALL_HOOK);
595         select_caller_info_hook(set_param_compare, ARRAY_LEN);
596         select_caller_info_hook(set_arraysize_arg, ARRAYSIZE_ARG);
597         add_hook(&munge_start_states, AFTER_DEF_HOOK);
598 }
599
600 void register_buf_comparison_links(int id)
601 {
602         link_id = id;
603         add_merge_hook(link_id, &merge_links);
604         add_modification_hook(link_id, &match_link_modify);
605 }
```

```
**********************************************************
   21671 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_buf_size.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include <errno.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"
#include "smatch_function_hashtable.h"

#define UNKNOWN_SIZE (-1)

static int my_size_id;

static DEFINE_HASHTABLE_INSERT(insert_func, char, int);
static DEFINE_HASHTABLE_SEARCH(search_func, char, int);
static struct hashtable *allocation_funcs;

static char *get_fn_name(struct expression *expr)
{
        if (expr->type != EXPR_CALL)
                return NULL;
        if (expr->fn->type != EXPR_SYMBOL)
                return NULL;
        return expr_to_var(expr->fn);
}

static int is_allocation_function(struct expression *expr)
{
        char *func;
        int ret = 0;

        func = get_fn_name(expr);
        if (!func)
                return 0;
        if (search_func(allocation_funcs, func))
                ret = 1;
        free_string(func);
        return ret;
}

static void add_allocation_function(const char *func, void *call_back, int param
{
        insert_func(allocation_funcs, (char *)func, (int *)1);
        add_function_assign_hook(func, call_back, INT_PTR(param));
```



```
}

static int estate_to_size(struct smatch_state *state)
{
        sval_t sval;

        if (!state || !estate_rl(state))
                return 0;
        sval = estate_max(state);
        return sval.value;
}

static struct smatch_state *size_to_estate(int size)
{
        sval_t sval;

        sval.type = &int_ctype;
        sval.value = size;

        return alloc_estate_sval(sval);
}

static struct range_list *size_to_rl(int size)
{
        sval_t sval;

        sval.type = &int_ctype;
        sval.value = size;

        return alloc_rl(sval, sval);
}

static struct smatch_state *unmatched_size_state(struct sm_state *sm)
{
        return size_to_estate(UNKNOWN_SIZE);
}

static void set_size_undefined(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(sm->owner, sm->name, sm->sym, size_to_estate(UNKNOWN_SIZE));
}

static struct smatch_state *merge_size_func(struct smatch_state *s1, struct smat
{
        return merge_estates(s1, s2);
}

void set_param_buf_size(const char *name, struct symbol *sym, char *key, char *v
{
        struct range_list *rl = NULL;
        struct smatch_state *state;
        char fullname[256];

        if (strncmp(key, "$", 1) != 0)
                return;

        snprintf(fullname, 256, "%s%s", name, key + 1);

        str_to_rl(&int_ctype, value, &rl);
        if (!rl || is_whole_rl(rl))
                return;
        state = alloc_estate_rl(rl);
        set_state(my_size_id, fullname, sym, state);
}

static int bytes_per_element(struct expression *expr)
```

```
{
        struct symbol *type;

        if (!expr)
                return 0;
        if (expr->type == EXPR_STRING)
                return 1;
        if (expr->type == EXPR_PREOP && expr->op == '&') {
                type = get_type(expr->unop);
                if (type && type->type == SYM_ARRAY)
                        expr = expr->unop;
        }
        type = get_type(expr);
        if (!type)
                return 0;

        if (type->type != SYM_PTR && type->type != SYM_ARRAY)
                return 0;

        type = get_base_type(type);
        return type_bytes(type);
}

static int bytes_to_elements(struct expression *expr, int bytes)
{
        int bpe;

        bpe = bytes_per_element(expr);
        if (bpe == 0)
                return 0;
        return bytes / bpe;
}

static int elements_to_bytes(struct expression *expr, int elements)
{
        int bpe;

        bpe = bytes_per_element(expr);
        return elements * bpe;
}

static int get_initializer_size(struct expression *expr)
{
        switch (expr->type) {
        case EXPR_STRING:
                return expr->string->length;
        case EXPR_INITIALIZER: {
                struct expression *tmp;
                int i = 0;

                FOR_EACH_PTR(expr->expr_list, tmp) {
                        if (tmp->type == EXPR_INDEX) {
                                if (tmp->idx_to >= i)
                                        i = tmp->idx_to;
                                else
                                        continue;
                        }

                        i++;
                } END_FOR_EACH_PTR(tmp);
                return i;
        }
        case EXPR_SYMBOL:
                return get_array_size(expr);
        }
        return 0;
```



```
}

static struct range_list *db_size_rl;
static int db_size_callback(void *unused, int argc, char **argv, char **azColNam
{
        struct range_list *tmp = NULL;

        if (!db_size_rl) {
                str_to_rl(&int_ctype, argv[0], &db_size_rl);
        } else {
                str_to_rl(&int_ctype, argv[0], &tmp);
                db_size_rl = rl_union(db_size_rl, tmp);
        }
        return 0;
}

static struct range_list *size_from_db_type(struct expression *expr)
{
        int this_file_only = 0;
        char *name;

        name = get_member_name(expr);
        if (!name && is_static(expr)) {
                name = expr_to_var(expr);
                this_file_only = 1;
        }
        if (!name)
                return NULL;

        if (this_file_only) {
                db_size_rl = NULL;
                run_sql(db_size_callback, NULL,
                        "select size from function_type_size where type = '%s' a
                        name, get_filename());
                if (db_size_rl)
                        return db_size_rl;
                return NULL;
        }

        db_size_rl = NULL;
        run_sql(db_size_callback, NULL,
                "select size from type_size where type = '%s';",
                name);
        return db_size_rl;
}

static struct range_list *size_from_db_symbol(struct expression *expr)
{
        struct symbol *sym;

        if (expr->type != EXPR_SYMBOL)
                return NULL;
        sym = expr->symbol;
        if (!sym || !sym->ident ||
            !(sym->ctype.modifiers & MOD_TOPLEVEL) ||
            sym->ctype.modifiers & MOD_STATIC)
                return NULL;

        db_size_rl = NULL;
        run_sql(db_size_callback, NULL,
                "select value from data_info where file = 'extern' and data = '%
                sym->ident->name, BUF_SIZE);
        return db_size_rl;
}

static struct range_list *size_from_db(struct expression *expr)
```

```
259 {
260         struct range_list *rl;

262         rl = size_from_db_symbol(expr);
263         if (rl)
264                 return rl;
265         return size_from_db_type(expr);
266 }

268 static void db_returns_buf_size(struct expression *expr, int param, char *unused
269 {
270         struct expression *call;
271         struct range_list *rl;

273         if (expr->type != EXPR_ASSIGNMENT)
274                 return;
275         call = strip_expr(expr->right);

277         if (!parse_call_math_rl(call, math, &rl))
278                 return;
279         rl = cast_rl(&int_ctype, rl);
280         set_state_expr(my_size_id, expr->left, alloc_estate_rl(rl));
281 }

283 static int get_real_array_size_from_type(struct symbol *type)
284 {
285         sval_t sval;

287         if (!type)
288                 return 0;
289         if (!type || type->type != SYM_ARRAY)
290                 return 0;

292         if (!get_implied_value(type->array_size, &sval))
293                 return 0;

295         return sval.value;
296 }

298 int get_real_array_size(struct expression *expr)
299 {
300         if (!expr)
301                 return 0;
302         if (expr->type == EXPR_PREOP && expr->op == '&')
303                 expr = expr->unop;
304         if (expr->type == EXPR_BINOP) /* array elements foo[5] */
305                 return 0;
306         return get_real_array_size_from_type(get_type(expr));
307 }

309 static int get_size_from_initializer(struct expression *expr)
310 {
311         if (expr->type != EXPR_SYMBOL || !expr->symbol || !expr->symbol->initial
312                 return 0;
313         if (expr->symbol->initializer == expr) /* int a = a; */
314                 return 0;
315         return get_initializer_size(expr->symbol->initializer);
316 }

318 static struct range_list *get_stored_size_bytes(struct expression *expr)
319 {
320         struct smatch_state *state;

322         state = get_state_expr(my_size_id, expr);
323         if (!state)
324                 return NULL;
```



```
325         return estate_rl(state);
326 }

328 static int get_bytes_from_address(struct expression *expr)
329 {
330         struct symbol *type;
331         int ret;

333         if (!option_spammy)
334                 return 0;
335         if (expr->type != EXPR_PREOP || expr->op != '&')
336                 return 0;
337         type = get_type(expr);
338         if (!type)
339                 return 0;

341         if (type->type == SYM_PTR)
342                 type = get_base_type(type);

344         ret = type_bytes(type);
345         if (ret == 1)
346                 return 0;  /* ignore char pointers */

348         return ret;
349 }

351 static struct expression *remove_addr_fluff(struct expression *expr)
352 {
353         struct expression *tmp;
354         sval_t sval;

356         expr = strip_expr(expr);

358         /* remove '&' and '*' operations that cancel */
359         while (expr && expr->type == EXPR_PREOP && expr->op == '&') {
360                 tmp = strip_expr(expr->unop);
361                 if (tmp->type != EXPR_PREOP)
362                         break;
363                 if (tmp->op != '*')
364                         break;
365                 expr = strip_expr(tmp->unop);
366         }

368         if (!expr)
369                 return NULL;

371         /* "foo + 0" is just "foo" */
372         if (expr->type == EXPR_BINOP && expr->op == '+' &&
373             get_value(expr->right, &sval) && sval.value == 0)
374                 return expr->left;

376         return expr;
377 }

379 static int is_last_member_of_struct(struct symbol *sym, struct ident *member)
380 {
381         struct symbol *tmp;
382         int i;

384         i = 0;
385         FOR_EACH_PTR_REVERSE(sym->symbol_list, tmp) {
386                 if (i++ || !tmp->ident)
387                         return 0;
388                 if (tmp->ident == member)
389                         return 1;
390                 return 0;
```

```
391         } END_FOR_EACH_PTR_REVERSE(tmp);

393         return 0;
394 }

396 int last_member_is_resizable(struct symbol *sym)
397 {
398         struct symbol *last_member;
399         struct symbol *type;
400         sval_t sval;

402         if (!sym || sym->type != SYM_STRUCT)
403                 return 0;

405         last_member = last_ptr_list((struct ptr_list *)sym->symbol_list);
406         if (!last_member || !last_member->ident)
407                 return 0;

409         type = get_real_base_type(last_member);
410         if (type->type == SYM_STRUCT)
411                 return last_member_is_resizable(type);
412         if (type->type != SYM_ARRAY)
413                 return 0;

415         if (!get_implied_value(type->array_size, &sval))
416                 return 0;

418         if (sval.value != 0 && sval.value != 1)
419                 return 0;

421         return 1;
422 }

424 static int get_stored_size_end_struct_bytes(struct expression *expr)
425 {
426         struct symbol *sym;
427         struct symbol *base_sym;
428         struct smatch_state *state;

430         if (expr->type == EXPR_BINOP) /* array elements foo[5] */
431                 return 0;

433         if (expr->type == EXPR_PREOP && expr->op == '&')
434                 expr = strip_parens(expr->unop);

436         sym = expr_to_sym(expr);
437         if (!sym || !sym->ident)
438                 return 0;
439         if (!type_bytes(sym))
440                 return 0;
441         if (sym->type != SYM_NODE)
442                 return 0;

444         base_sym = get_real_base_type(sym);
445         if (!base_sym || base_sym->type != SYM_PTR)
446                 return 0;
447         base_sym = get_real_base_type(base_sym);
448         if (!base_sym || base_sym->type != SYM_STRUCT)
449                 return 0;

451         if (!is_last_member_of_struct(base_sym, expr->member))
452                 return 0;
453         if (!last_member_is_resizable(base_sym))
454                 return 0;

456         state = get_state(my_size_id, sym->ident->name, sym);
```



```
457         if (!estate_to_size(state))
458                 return 0;

460         return estate_to_size(state) - type_bytes(base_sym) + type_bytes(get_typ
461 }

463 static struct range_list *alloc_int_rl(int value)
464 {
465         sval_t sval = {
466                 .type = &int_ctype,
467                 {.value = value},
468         };

470         return alloc_rl(sval, sval);
471 }

473 struct range_list *get_array_size_bytes_rl(struct expression *expr)
474 {
475         struct range_list *ret = NULL;
476         int size;

478         expr = remove_addr_fluff(expr);
479         if (!expr)
480                 return NULL;

482         /* "BAR" */
483         if (expr->type == EXPR_STRING)
484                 return alloc_int_rl(expr->string->length);

486         if (expr->type == EXPR_BINOP && expr->op == '+') {
487                 sval_t offset;
488                 struct symbol *type;
489                 int bytes;

491                 if (!get_implied_value(expr->right, &offset))
492                         return NULL;
493                 type = get_type(expr->left);
494                 if (!type)
495                         return NULL;
496                 if (type->type != SYM_ARRAY && type->type != SYM_PTR)
497                         return NULL;
498                 type = get_real_base_type(type);
499                 bytes = type_bytes(type);
500                 if (bytes == 0)
501                         return NULL;
502                 offset.value *= bytes;
503                 size = get_array_size_bytes(expr->left);
504                 if (size <= 0)
505                         return NULL;
506                 return alloc_int_rl(size - offset.value);
507         }

509         /* buf[4] */
510         size = get_real_array_size(expr);
511         if (size)
512                 return alloc_int_rl(elements_to_bytes(expr, size));

514         /* buf = malloc(1024); */
515         ret = get_stored_size_bytes(expr);
516         if (ret)
517                 return ret;

519         size = get_stored_size_end_struct_bytes(expr);
520         if (size)
521                 return alloc_int_rl(size);
```

```
523         /* char *foo = "BAR" */
524         size = get_size_from_initializer(expr);
525         if (size)
526                 return alloc_int_rl(elements_to_bytes(expr, size));

528         size = get_bytes_from_address(expr);
529         if (size)
530                 return alloc_int_rl(size);

532         ret = size_from_db(expr);
533         if (ret)
534                 return ret;

536         return NULL;
537 }

539 int get_array_size_bytes(struct expression *expr)
540 {
541         struct range_list *rl;
542         sval_t sval;

544         rl = get_array_size_bytes_rl(expr);
545         if (!rl_to_sval(rl, &sval))
546                 return 0;
547         if (sval.uvalue >= INT_MAX)
548                 return 0;
549         return sval.value;
550 }

552 int get_array_size_bytes_max(struct expression *expr)
553 {
554         struct range_list *rl;
555         sval_t bytes;

557         rl = get_array_size_bytes_rl(expr);
558         if (!rl)
559                 return 0;
560         bytes = rl_min(rl);
561         if (bytes.value < 0)
562                 return 0;
563         bytes = rl_max(rl);
564         if (bytes.uvalue >= INT_MAX)
565                 return 0;
566         return bytes.value;
567 }

569 int get_array_size_bytes_min(struct expression *expr)
570 {
571         struct range_list *rl;
572         struct data_range *range;

574         rl = get_array_size_bytes_rl(expr);
575         if (!rl)
576                 return 0;

578         FOR_EACH_PTR(rl, range) {
579                 if (range->min.value <= 0)
580                         return 0;
581                 if (range->max.value <= 0)
582                         return 0;
583                 if (range->min.uvalue >= INT_MAX)
584                         return 0;
585                 return range->min.value;
586         } END_FOR_EACH_PTR(range);

588         return 0;
```



```
589 }

591 int get_array_size(struct expression *expr)
592 {
593         if (!expr)
594                 return 0;
595         return bytes_to_elements(expr, get_array_size_bytes_max(expr));
596 }

598 static struct expression *strip_ampersands(struct expression *expr)
599 {
600         struct symbol *type;

602         if (expr->type != EXPR_PREOP)
603                 return expr;
604         if (expr->op != '&')
605                 return expr;
606         type = get_type(expr->unop);
607         if (!type || type->type != SYM_ARRAY)
608                 return expr;
609         return expr->unop;
610 }

612 static void info_record_alloction(struct expression *buffer, struct range_list *
613 {
614         char *name;

616         if (!option_info)
617                 return;

619         name = get_member_name(buffer);
620         if (!name && is_static(buffer))
621                 name = expr_to_var(buffer);
622         if (!name)
623                 return;
624         if (rl && !is_whole_rl(rl))
625                 sql_insert_function_type_size(name, show_rl(rl));
626         else
627                 sql_insert_function_type_size(name, "(-1)");

629         free_string(name);
630 }

632 static void store_alloc(struct expression *expr, struct range_list *rl)
633 {
634         struct symbol *type;

636         rl = clone_rl(rl); // FIXME!!!
637         set_state_expr(my_size_id, expr, alloc_estate_rl(rl));

639         type = get_type(expr);
640         if (!type)
641                 return;
642         if (type->type != SYM_PTR)
643                 return;
644         type = get_real_base_type(type);
645         if (!type)
646                 return;
647         if (type == &void_ctype)
648                 return;
649         if (type->type != SYM_BASETYPE && type->type != SYM_PTR)
650                 return;

652         info_record_alloction(expr, rl);
653 }
```

```
655 static void match_array_assignment(struct expression *expr)
656 {
657         struct expression *left;
658         struct expression *right;
659         char *left_member, *right_member;
660         struct range_list *rl;
661         sval_t sval;
662
663         if (expr->op != '=')
664                 return;
665         left = strip_expr(expr->left);
666         right = strip_expr(expr->right);
667         right = strip_ampersands(right);
668
669         if (!is_pointer(left))
670                 return;
671         if (is_allocation_function(right))
672                 return;
673
674         left_member = get_member_name(left);
675         right_member = get_member_name(right);
676         if (left_member && right_member && strcmp(left_member, right_member) ==
677                 free_string(left_member);
678                 free_string(right_member);
679                 return;
680         }
681         free_string(left_member);
682         free_string(right_member);
683
684         if (get_implied_value(right, &sval) && sval.value == 0) {
685                 rl = alloc_int_rl(0);
686                 goto store;
687         }
688
689         rl = get_array_size_bytes_rl(right);
690         if (!rl && __in_fake_assign)
691                 return;
692
693 store:
694         store_alloc(left, rl);
695 }
696
697 static void match_alloc(const char *fn, struct expression *expr, void *_size_arg
698 {
699         int size_arg = PTR_INT(_size_arg);
700         struct expression *right;
701         struct expression *arg;
702         struct range_list *rl;
703
704         right = strip_expr(expr->right);
705         arg = get_argument_from_call_expr(right->args, size_arg);
706         get_absolute_rl(arg, &rl);
707         rl = cast_rl(&int_ctype, rl);
708         store_alloc(expr->left, rl);
709 }
710
711 static void match_calloc(const char *fn, struct expression *expr, void *unused)
712 {
713         struct expression *right;
714         struct expression *arg;
715         sval_t elements;
716         sval_t size;
717
718         right = strip_expr(expr->right);
719         arg = get_argument_from_call_expr(right->args, 0);
720         if (!get_implied_value(arg, &elements))
```



```
721                 return; // FIXME!!!
722         arg = get_argument_from_call_expr(right->args, 1);
723         if (get_implied_value(arg, &size))
724                 store_alloc(expr->left, size_to_rl(elements.value * size.value))
725         else
726                 store_alloc(expr->left, size_to_rl(-1));
727 }
728
729 static void match_page(const char *fn, struct expression *expr, void *_unused)
730 {
731         sval_t page_size = {
732                 .type = &int_ctype,
733                 {.value = 4096},
734         };
735
736         store_alloc(expr->left, alloc_rl(page_size, page_size));
737 }
738
739 static void match_strndup(const char *fn, struct expression *expr, void *unused)
740 {
741         struct expression *fn_expr;
742         struct expression *size_expr;
743         sval_t size;
744
745         fn_expr = strip_expr(expr->right);
746         size_expr = get_argument_from_call_expr(fn_expr->args, 1);
747         if (get_implied_max(size_expr, &size)) {
748                 size.value++;
749                 store_alloc(expr->left, size_to_rl(size.value));
750         } else {
751                 store_alloc(expr->left, size_to_rl(-1));
752         }
753
754 }
755
756 static void match_alloc_pages(const char *fn, struct expression *expr, void *_or
757 {
758         int order_arg = PTR_INT(_order_arg);
759         struct expression *right;
760         struct expression *arg;
761         sval_t sval;
762
763         right = strip_expr(expr->right);
764         arg = get_argument_from_call_expr(right->args, order_arg);
765         if (!get_implied_value(arg, &sval))
766                 return;
767         if (sval.value < 0 || sval.value > 10)
768                 return;
769
770         sval.type = &int_ctype;
771         sval.value = 1 << sval.value;
772         sval.value *= 4096;
773
774         store_alloc(expr->left, alloc_rl(sval, sval));
775 }
776
777 static int is_type_bytes(struct range_list *rl, struct expression *arg)
778 {
779         struct symbol *type;
780         sval_t sval;
781
782         if (!rl_to_sval(rl, &sval))
783                 return 0;
784
785         type = get_type(arg);
786         if (!type)
```

```
787                 return 0;
788         if (type->type != SYM_PTR)
789                 return 0;
790         type = get_real_base_type(type);
791         if (type->type != SYM_STRUCT &&
792             type->type != SYM_UNION)
793                 return 0;
794         if (sval.value != type_bytes(type))
795                 return 0;
796         return 1;
797 }

799 static void match_call(struct expression *expr)
800 {
801         struct expression *arg;
802         struct symbol *type;
803         struct range_list *rl;
804         int i;

806         i = -1;
807         FOR_EACH_PTR(expr->args, arg) {
808                 i++;
809                 type = get_type(arg);
810                 if (!type || (type->type != SYM_PTR && type->type != SYM_ARRAY))
811                         continue;
812                 rl = get_array_size_bytes_rl(arg);
813                 if (!rl)
814                         continue;
815                 if (is_whole_rl(rl))
816                         continue;
817                 if (is_type_bytes(rl, arg))
818                         continue;
819                 sql_insert_caller_info(expr, BUF_SIZE, i, "$", show_rl(rl));
820         } END_FOR_EACH_PTR(arg);
821 }

823 static void struct_member_callback(struct expression *call, int param, char *pri
824 {
825         if (sm->state == &merged ||
826             strcmp(sm->state->name, "(-1)") == 0 ||
827             strcmp(sm->state->name, "empty") == 0 ||
828             strcmp(sm->state->name, "0") == 0)
829                 return;
830         sql_insert_caller_info(call, BUF_SIZE, param, printed_name, sm->state->n
831 }

833 /*
834  * This is slightly (very) weird because half of this stuff is handled in
835  * smatch_parse_call_math.c which is poorly named.  But anyway, add some buf
836  * sizes here.
837  *
838  */
839 static void print_returned_allocations(int return_id, char *return_ranges, struc
840 {
841         char buf[16];
842         int size;

844         size = get_array_size_bytes(expr);
845         if (!size)
846                 return;

848         snprintf(buf, sizeof(buf), "%d", size);
849         sql_insert_return_states(return_id, return_ranges, BUF_SIZE, -1, "", buf
850 }

852 static void record_global_size(struct symbol *sym)
```



```
853 {
854         int bytes;
855         char buf[16];

857         if (!sym->ident)
858                 return;

860         if (!(sym->ctype.modifiers & MOD_TOPLEVEL) ||
861             sym->ctype.modifiers & MOD_STATIC)
862                 return;

864         bytes = get_array_size_bytes(symbol_expression(sym));
865         if (bytes <= 1)
866                 return;

868         snprintf(buf, sizeof(buf), "%d", bytes);
869         sql_insert_data_info_var_sym(sym->ident->name, sym, BUF_SIZE, buf);
870 }

872 void register_buf_size(int id)
873 {
874         my_size_id = id;

876         add_unmatched_state_hook(my_size_id, &unmatched_size_state);

878         select_caller_info_hook(set_param_buf_size, BUF_SIZE);
879         select_return_states_hook(BUF_SIZE, &db_returns_buf_size);
880         add_split_return_callback(print_returned_allocations);

882         allocation_funcs = create_function_hashtable(100);
883         add_allocation_function("malloc", &match_alloc, 0);
884         add_allocation_function("calloc", &match_calloc, 0);
885         add_allocation_function("memdup", &match_alloc, 1);
886         add_allocation_function("realloc", &match_alloc, 1);
887         if (option_project == PROJ_KERNEL) {
888                 add_allocation_function("kmalloc", &match_alloc, 0);
889                 add_allocation_function("kmalloc_node", &match_alloc, 0);
890                 add_allocation_function("kzalloc", &match_alloc, 0);
891                 add_allocation_function("kzalloc_node", &match_alloc, 0);
892                 add_allocation_function("vmalloc", &match_alloc, 0);
893                 add_allocation_function("__vmalloc", &match_alloc, 0);
894                 add_allocation_function("kcalloc", &match_calloc, 0);
895                 add_allocation_function("kmalloc_array", &match_calloc, 0);
896                 add_allocation_function("drm_malloc_ab", &match_calloc, 0);
897                 add_allocation_function("drm_calloc_large", &match_calloc, 0);
898                 add_allocation_function("sock_kmalloc", &match_alloc, 1);
899                 add_allocation_function("kmemdup", &match_alloc, 1);
900                 add_allocation_function("kmemdup_user", &match_alloc, 1);
901                 add_allocation_function("dma_alloc_attrs", &match_alloc, 1);
902                 add_allocation_function("pci_alloc_consistent", &match_alloc, 1)
903                 add_allocation_function("pci_alloc_coherent", &match_alloc, 1);
904                 add_allocation_function("devm_kmalloc", &match_alloc, 1);
905                 add_allocation_function("devm_kzalloc", &match_alloc, 1);
906                 add_allocation_function("krealloc", &match_alloc, 1);
907                 add_allocation_function("__alloc_bootmem", &match_alloc, 0);
908                 add_allocation_function("alloc_bootmem", &match_alloc, 0);
909                 add_allocation_function("kmap", &match_page, 0);
910                 add_allocation_function("get_zeroed_page", &match_page, 0);
911                 add_allocation_function("alloc_pages", &match_alloc_pages, 1);
912                 add_allocation_function("alloc_pages_current", &match_alloc_page
913                 add_allocation_function("__get_free_pages", &match_alloc_pages,
914         }

916         add_allocation_function("strndup", match_strndup, 0);
917         if (option_project == PROJ_KERNEL)
918                 add_allocation_function("kstrndup", match_strndup, 0);
```

```
920         add_modification_hook(my_size_id, &set_size_undefined);

922         add_merge_hook(my_size_id, &merge_size_func);

924         if (option_info)
925                 add_hook(record_global_size, BASE_HOOK);
926 }

928 void register_buf_size_late(int id)
929 {
930         /* has to happen after match_alloc() */
931         add_hook(&match_array_assignment, ASSIGNMENT_HOOK);

933         add_hook(&match_call, FUNCTION_CALL_HOOK);
934         add_member_info_callback(my_size_id, struct_member_callback);
935 }
```

```
**********************************************************
    7360 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_capped.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Oracle.  All rights reserved.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is trying to make a list of the variables which
 * have capped values.  Sometimes we don't know what the
 * cap is, for example if we are comparing variables but
 * we don't know the values of the variables.  In that
 * case we only know that our variable is capped and we
 * sort that information here.
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(capped);
STATE(uncapped);

static void set_uncapped(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, &uncapped);
}

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        struct smatch_state *state;

        state = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (state && !estate_is_whole(state))
                return &capped;
        return &uncapped;
}

static int is_capped_macro(struct expression *expr)
{
        char *name;

        name = get_macro_name(expr->pos);
        if (!name)
                return 0;

        if (strcmp(name, "min") == 0)
                return 1;
```



```
        if (strcmp(name, "MIN") == 0)
                return 1;
        if (strcmp(name, "min_t") == 0)
                return 1;

        return 0;
}

int is_capped(struct expression *expr)
{
        sval_t dummy;

        expr = strip_expr(expr);
        while (expr && expr->type == EXPR_POSTOP) {
                expr = strip_expr(expr->unop);
        }
        if (!expr)
                return 0;

        if (get_hard_max(expr, &dummy))
                return 1;

        if (is_capped_macro(expr))
                return 1;

        if (expr->type == EXPR_BINOP) {
                struct range_list *left_rl, *right_rl;

                if (expr->op == '&')
                        return 1;
                if (expr->op == SPECIAL_RIGHTSHIFT)
                        return 1;
                if (expr->op == '%')
                        return is_capped(expr->right);
                if (!is_capped(expr->left))
                        return 0;
                if (expr->op == '/')
                        return 1;
                if (!is_capped(expr->right))
                        return 0;
                if (expr->op == '*') {
                        get_absolute_rl(expr->left, &left_rl);
                        get_absolute_rl(expr->right, &right_rl);
                        if (sval_is_negative(rl_min(left_rl)) ||
                            sval_is_negative(rl_min(right_rl)))
                                return 0;
                }
                return 1;
        }
        if (get_state_expr(my_id, expr) == &capped)
                return 1;
        return 0;
}

int is_capped_var_sym(const char *name, struct symbol *sym)
{
        if (get_state(my_id, name, sym) == &capped)
                return 1;
        return 0;
}

void set_param_capped_data(const char *name, struct symbol *sym, char *key, char
{
        char fullname[256];

        if (strncmp(key, "$", 1))
```

```
127                 return;
128         snprintf(fullname, 256, "%s%s", name, key + 1);
129         set_state(my_id, fullname, sym, &capped);
130 }

132 static void match_condition(struct expression *expr)
133 {
134         struct smatch_state *left_true = NULL;
135         struct smatch_state *left_false = NULL;
136         struct smatch_state *right_true = NULL;
137         struct smatch_state *right_false = NULL;


140         if (expr->type != EXPR_COMPARE)
141                 return;

143         switch (expr->op) {
144         case '<':
145         case SPECIAL_LTE:
146         case SPECIAL_UNSIGNED_LT:
147         case SPECIAL_UNSIGNED_LTE:
148                 left_true = &capped;
149                 right_false = &capped;
150                 break;
151         case '>':
152         case SPECIAL_GTE:
153         case SPECIAL_UNSIGNED_GT:
154         case SPECIAL_UNSIGNED_GTE:
155                 left_false = &capped;
156                 right_true = &capped;
157                 break;
158         case SPECIAL_EQUAL:
159                 left_true = &capped;
160                 right_true = &capped;
161                 break;
162         case SPECIAL_NOTEQUAL:
163                 left_false = &capped;
164                 right_false = &capped;
165                 break;

167         default:
168                 return;
169         }

171         set_true_false_states_expr(my_id, expr->left, left_true, left_false);
172         set_true_false_states_expr(my_id, expr->right, right_true, right_false);
173 }

175 static void match_assign(struct expression *expr)
176 {
177         if (is_capped(expr->right)) {
178                 set_state_expr(my_id, expr->left, &capped);
179         } else {
180                 if (get_state_expr(my_id, expr->left))
181                         set_state_expr(my_id, expr->left, &uncapped);
182         }
183 }

185 static void match_caller_info(struct expression *expr)
186 {
187         struct expression *tmp;
188         sval_t sval;
189         int i;

191         i = -1;
192         FOR_EACH_PTR(expr->args, tmp) {
```



```
193                 i++;
194                 if (get_implied_value(tmp, &sval))
195                         continue;
196                 if (!is_capped(tmp))
197                         continue;
198                 sql_insert_caller_info(expr, CAPPED_DATA, i, "$", "1");
199         } END_FOR_EACH_PTR(tmp);
200 }

202 static void struct_member_callback(struct expression *call, int param, char *pri
203 {
204         struct smatch_state *estate;
205         sval_t sval;

207         if (sm->state != &capped)
208                 return;
209         estate = get_state(SMATCH_EXTRA, sm->name, sm->sym);
210         if (estate_get_single_value(estate, &sval))
211                 return;
212         sql_insert_caller_info(call, CAPPED_DATA, param, printed_name, "1");
213 }

215 static void print_return_implies_capped(int return_id, char *return_ranges, stru
216 {
217         struct smatch_state *orig, *estate;
218         struct sm_state *sm;
219         struct symbol *ret_sym;
220         const char *param_name;
221         char *return_str;
222         int param;
223         sval_t sval;
224         bool return_found = false;

226         expr = strip_expr(expr);
227         return_str = expr_to_str(expr);
228         ret_sym = expr_to_sym(expr);

230         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
231                 if (sm->state != &capped)
232                         continue;

234                 param = get_param_num_from_sym(sm->sym);
235                 if (param < 0)
236                         continue;

238                 estate = get_state(SMATCH_EXTRA, sm->name, sm->sym);
239                 if (estate_get_single_value(estate, &sval))
240                         continue;

242                 orig = get_state_stree(get_start_states(), my_id, sm->name, sm->
243                 if (orig == &capped)
244                         continue;

246                 param_name = get_param_name(sm);
247                 if (!param_name)
248                         continue;

250                 sql_insert_return_states(return_id, return_ranges, CAPPED_DATA,
251                                          param, param_name, "1");
252         } END_FOR_EACH_SM(sm);

254         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
255                 if (!ret_sym)
256                         break;
257                 if (ret_sym != sm->sym)
258                         continue;
```

```
260                 param_name = state_name_to_param_name(sm->name, return_str);
261                 if (!param_name)
262                         continue;
263                 if (strcmp(param_name, "$") == 0)
264                         return_found = true;
265                 sql_insert_return_states(return_id, return_ranges, CAPPED_DATA,
266                                          -1, param_name, "1");
267         } END_FOR_EACH_SM(sm);

269         if (return_found)
270                 goto free_string;

272         if (option_project == PROJ_KERNEL && get_function() &&
273             strstr(get_function(), "nla_get_"))
274                 sql_insert_return_states(return_id, return_ranges, CAPPED_DATA,
275                                          -1, "$", "1");

277 free_string:
278         free_string(return_str);
279 }

281 static void db_return_states_capped(struct expression *expr, int param, char *ke
282 {
283         char *name;
284         struct symbol *sym;

286         name = return_state_to_var_sym(expr, param, key, &sym);
287         if (!name || !sym)
288                 goto free;

290         set_state(my_id, name, sym, &capped);
291 free:
292         free_string(name);
293 }

295 void register_capped(int id)
296 {
297         my_id = id;

299         add_unmatched_state_hook(my_id, &unmatched_state);
300         select_caller_info_hook(set_param_capped_data, CAPPED_DATA);
301         add_hook(&match_condition, CONDITION_HOOK);
302         add_hook(&match_assign, ASSIGNMENT_HOOK);
303         add_modification_hook(my_id, &set_uncapped);

305         add_hook(&match_caller_info, FUNCTION_CALL_HOOK);
306         add_member_info_callback(my_id, struct_member_callback);

308         add_split_return_callback(print_return_implies_capped);
309         select_return_states_hook(CAPPED_DATA, &db_return_states_capped);
310 }
```

```
**********************************************************
    2838 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_common_functions.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_extra.h"

static int match_strlen(struct expression *call, void *unused, struct range_list
{
        struct expression *str;
        unsigned long max;

        str = get_argument_from_call_expr(call->args, 0);
        if (get_implied_strlen(str, rl) && sval_is_positive(rl_min(*rl))) {
                *rl = cast_rl(&ulong_ctype, *rl);
                return 1;
        }
        /* smatch_strlen.c is not very complete */
        max = get_array_size_bytes_max(str);
        if (max == 0) {
                *rl = alloc_rl(sval_type_val(&ulong_ctype, 0),
                               sval_type_val(&ulong_ctype, STRLEN_MAX_RET));
        } else {
                max--;
                *rl = alloc_rl(sval_type_val(&ulong_ctype, 0),
                               sval_type_val(&ulong_ctype, max));
        }
        return 1;
}

static int match_strnlen(struct expression *call, void *unused, struct range_lis
{
        struct expression *limit;
        sval_t fixed;
        sval_t bound;
        sval_t ulong_max = sval_type_val(&ulong_ctype, ULONG_MAX);

        match_strlen(call, NULL, rl);
        limit = get_argument_from_call_expr(call->args, 1);
        if (!get_implied_max(limit, &bound))
                return 1;
        if (sval_cmp(bound, ulong_max) == 0)
                return 1;
        if (rl_to_sval(*rl, &fixed) && sval_cmp(fixed, bound) >= 0) {
                *rl = alloc_rl(bound, bound);
                return 1;
```



```
        }

        bound.value++;
        *rl = remove_range(*rl, bound, ulong_max);

        return 1;
}

static int match_sprintf(struct expression *call, void *_arg, struct range_list
{
        int str_arg = PTR_INT(_arg);
        int size;

        size = get_formatted_string_size(call, str_arg);
        if (size <= 0) {
                *rl = alloc_whole_rl(&ulong_ctype);
        } else {
                /* FIXME:  This is bogus.  get_formatted_string_size() should be
                   returning a range_list.  Also it should not add the NUL. */
                size--;
                *rl = alloc_rl(ll_to_sval(0), ll_to_sval(size));
        }
        return 1;
}

void register_common_functions(int id)
{
        /*
         * When you add a new function here, then don't forget to delete it from
         * the database and smatch_data/.
         */
        add_implied_return_hook("strlen", &match_strlen, NULL);
        add_implied_return_hook("strnlen", &match_strnlen, NULL);
        add_implied_return_hook("sprintf", &match_sprintf, INT_PTR(1));
        add_implied_return_hook("snprintf", &match_sprintf, INT_PTR(2));
}
```

```
**********************************************************
   61652 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_comparison.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The point here is to store the relationships between two variables.
 * Ie:  y > x.
 * To do that we create a state with the two variables in alphabetical order:
 * ->name = "x vs y" and the state would be "<".  On the false path the state
 * would be ">=".
 *
 * Part of the trick of it is that if x or y is modified then we need to reset
 * the state.  We need to keep a list of all the states which depend on x and
 * all the states which depend on y.  The link_id code handles this.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int compare_id;
static int link_id;
static int inc_dec_id;
static int inc_dec_link_id;

static void add_comparison(struct expression *left, int comparison, struct expre

/* for handling for loops */
STATE(start);
STATE(incremented);

ALLOCATOR(compare_data, "compare data");

static struct symbol *vsl_to_sym(struct var_sym_list *vsl)
{
        struct var_sym *vs;

        if (!vsl)
                return NULL;
        if (ptr_list_size((struct ptr_list *)vsl) != 1)
                return NULL;
        vs = first_ptr_list((struct ptr_list *)vsl);
        return vs->sym;
}

struct smatch_state *alloc_compare_state(
```



```
                struct expression *left,
                const char *left_var, struct var_sym_list *left_vsl,
                int comparison,
                struct expression *right,
                const char *right_var, struct var_sym_list *right_vsl)
{
        struct smatch_state *state;
        struct compare_data *data;

        state = __alloc_smatch_state(0);
        state->name = alloc_sname(show_special(comparison));
        data = __alloc_compare_data(0);
        data->left = left;
        data->left_var = alloc_sname(left_var);
        data->left_vsl = clone_var_sym_list(left_vsl);
        data->comparison = comparison;
        data->right = right;
        data->right_var = alloc_sname(right_var);
        data->right_vsl = clone_var_sym_list(right_vsl);
        state->data = data;
        return state;
}

int state_to_comparison(struct smatch_state *state)
{
        if (!state || !state->data)
                return 0;
        return ((struct compare_data *)state->data)->comparison;
}

/*
 * flip_comparison() reverses the op left and right.  So "x >= y" becomes "y <=
 */
int flip_comparison(int op)
{
        switch (op) {
        case 0:
                return 0;
        case '<':
                return '>';
        case SPECIAL_UNSIGNED_LT:
                return SPECIAL_UNSIGNED_GT;
        case SPECIAL_LTE:
                return SPECIAL_GTE;
        case SPECIAL_UNSIGNED_LTE:
                return SPECIAL_UNSIGNED_GTE;
        case SPECIAL_EQUAL:
                return SPECIAL_EQUAL;
        case SPECIAL_NOTEQUAL:
                return SPECIAL_NOTEQUAL;
        case SPECIAL_GTE:
                return SPECIAL_LTE;
        case SPECIAL_UNSIGNED_GTE:
                return SPECIAL_UNSIGNED_LTE;
        case '>':
                return '<';
        case SPECIAL_UNSIGNED_GT:
                return SPECIAL_UNSIGNED_LT;
        default:
                sm_perror("unhandled comparison %d", op);
                return op;
        }
}

int negate_comparison(int op)
{
```

```
127         switch (op) {
128         case 0:
129                 return 0;
130         case '<':
131                 return SPECIAL_GTE;
132         case SPECIAL_UNSIGNED_LT:
133                 return SPECIAL_UNSIGNED_GTE;
134         case SPECIAL_LTE:
135                 return '>';
136         case SPECIAL_UNSIGNED_LTE:
137                 return SPECIAL_UNSIGNED_GT;
138         case SPECIAL_EQUAL:
139                 return SPECIAL_NOTEQUAL;
140         case SPECIAL_NOTEQUAL:
141                 return SPECIAL_EQUAL;
142         case SPECIAL_GTE:
143                 return '<';
144         case SPECIAL_UNSIGNED_GTE:
145                 return SPECIAL_UNSIGNED_LT;
146         case '>':
147                 return SPECIAL_LTE;
148         case SPECIAL_UNSIGNED_GT:
149                 return SPECIAL_UNSIGNED_LTE;
150         default:
151                 sm_perror("unhandled comparison %d", op);
152                 return op;
153         }
154 }

156 static int rl_comparison(struct range_list *left_rl, struct range_list *right_rl
157 {
158         sval_t left_min, left_max, right_min, right_max;
159         struct symbol *type = &int_ctype;

161         if (!left_rl || !right_rl)
162                 return 0;

164         if (type_positive_bits(rl_type(left_rl)) > type_positive_bits(type))
165                 type = rl_type(left_rl);
166         if (type_positive_bits(rl_type(right_rl)) > type_positive_bits(type))
167                 type = rl_type(right_rl);

169         left_rl = cast_rl(type, left_rl);
170         right_rl = cast_rl(type, right_rl);

172         left_min = rl_min(left_rl);
173         left_max = rl_max(left_rl);
174         right_min = rl_min(right_rl);
175         right_max = rl_max(right_rl);

177         if (left_min.value == left_max.value &&
178             right_min.value == right_max.value &&
179             left_min.value == right_min.value)
180                 return SPECIAL_EQUAL;

182         if (sval_cmp(left_max, right_min) < 0)
183                 return '<';
184         if (sval_cmp(left_max, right_min) == 0)
185                 return SPECIAL_LTE;
186         if (sval_cmp(left_min, right_max) > 0)
187                 return '>';
188         if (sval_cmp(left_min, right_max) == 0)
189                 return SPECIAL_GTE;

191         return 0;
192 }
```



```
194 static int comparison_from_extra(struct expression *a, struct expression *b)
195 {
196         struct range_list *left, *right;

198         if (!get_implied_rl(a, &left))
199                 return 0;
200         if (!get_implied_rl(b, &right))
201                 return 0;

203         return rl_comparison(left, right);
204 }

206 static struct range_list *get_orig_rl(struct var_sym_list *vsl)
207 {
208         struct symbol *sym;
209         struct smatch_state *state;

211         if (!vsl)
212                 return NULL;
213         sym = vsl_to_sym(vsl);
214         if (!sym || !sym->ident)
215                 return NULL;
216         state = get_orig_estate(sym->ident->name, sym);
217         return estate_rl(state);
218 }

220 static struct smatch_state *unmatched_comparison(struct sm_state *sm)
221 {
222         struct compare_data *data = sm->state->data;
223         struct range_list *left_rl, *right_rl;
224         int op;

226         if (!data)
227                 return &undefined;

229         if (strstr(data->left_var, " orig"))
230                 left_rl = get_orig_rl(data->left_vsl);
231         else if (!get_implied_rl_var_sym(data->left_var, vsl_to_sym(data->left_v
232                 return &undefined;

234         if (strstr(data->right_var, " orig"))
235                 right_rl = get_orig_rl(data->right_vsl);
236         else if (!get_implied_rl_var_sym(data->right_var, vsl_to_sym(data->right
237                 return &undefined;

239         op = rl_comparison(left_rl, right_rl);
240         if (op)
241                 return alloc_compare_state(
242                                 data->left, data->left_var, data->left_vsl,
243                                 op,
244                                 data->right, data->right_var, data->right_vsl);

246         return &undefined;
247 }

249 /* remove_unsigned_from_comparison() is obviously a hack. */
250 int remove_unsigned_from_comparison(int op)
251 {
252         switch (op) {
253         case SPECIAL_UNSIGNED_LT:
254                 return '<';
255         case SPECIAL_UNSIGNED_LTE:
256                 return SPECIAL_LTE;
257         case SPECIAL_UNSIGNED_GTE:
258                 return SPECIAL_GTE;
```

```
259         case SPECIAL_UNSIGNED_GT:
260                 return '>';
261         default:
262                 return op;
263         }
264 }

266 /*
267  * This is for when you merge states "a < b" and "a == b", the result is that
268  * we can say for sure, "a <= b" after the merge.
269  */
270 int merge_comparisons(int one, int two)
271 {
272         int LT, EQ, GT;

274         if (!one || !two)
275                 return 0;

277         one = remove_unsigned_from_comparison(one);
278         two = remove_unsigned_from_comparison(two);

280         if (one == two)
281                 return one;

283         LT = EQ = GT = 0;

285         switch (one) {
286         case '<':
287                 LT = 1;
288                 break;
289         case SPECIAL_LTE:
290                 LT = 1;
291                 EQ = 1;
292                 break;
293         case SPECIAL_EQUAL:
294                 EQ = 1;
295                 break;
296         case SPECIAL_GTE:
297                 GT = 1;
298                 EQ = 1;
299                 break;
300         case '>':
301                 GT = 1;
302         }

304         switch (two) {
305         case '<':
306                 LT = 1;
307                 break;
308         case SPECIAL_LTE:
309                 LT = 1;
310                 EQ = 1;
311                 break;
312         case SPECIAL_EQUAL:
313                 EQ = 1;
314                 break;
315         case SPECIAL_GTE:
316                 GT = 1;
317                 EQ = 1;
318                 break;
319         case '>':
320                 GT = 1;
321         }

323         if (LT && EQ && GT)
324                 return 0;
```



```
325         if (LT && EQ)
326                 return SPECIAL_LTE;
327         if (LT && GT)
328                 return SPECIAL_NOTEQUAL;
329         if (LT)
330                 return '<';
331         if (EQ && GT)
332                 return SPECIAL_GTE;
333         if (GT)
334                 return '>';
335         return 0;
336 }

338 /*
339  * This is for if you have "a < b" and "b <= c".  Or in other words,
340  * "a < b <= c".  You would call this like get_combined_comparison('<', '<=').
341  * The return comparison would be '<'.
342  *
343  * This function is different from merge_comparisons(), for example:
344  * merge_comparison('<', '==') returns '<='
345  * get_combined_comparison('<', '==') returns '<'
346  */
347 int combine_comparisons(int left_compare, int right_compare)
348 {
349         int LT, EQ, GT;

351         left_compare = remove_unsigned_from_comparison(left_compare);
352         right_compare = remove_unsigned_from_comparison(right_compare);

354         LT = EQ = GT = 0;

356         switch (left_compare) {
357         case '<':
358                 LT++;
359                 break;
360         case SPECIAL_LTE:
361                 LT++;
362                 EQ++;
363                 break;
364         case SPECIAL_EQUAL:
365                 return right_compare;
366         case SPECIAL_GTE:
367                 GT++;
368                 EQ++;
369                 break;
370         case '>':
371                 GT++;
372         }

374         switch (right_compare) {
375         case '<':
376                 LT++;
377                 break;
378         case SPECIAL_LTE:
379                 LT++;
380                 EQ++;
381                 break;
382         case SPECIAL_EQUAL:
383                 return left_compare;
384         case SPECIAL_GTE:
385                 GT++;
386                 EQ++;
387                 break;
388         case '>':
389                 GT++;
390         }
```

```
392         if (LT == 2) {
393                 if (EQ == 2)
394                         return SPECIAL_LTE;
395                 return '<';
396         }

398         if (GT == 2) {
399                 if (EQ == 2)
400                         return SPECIAL_GTE;
401                 return '>';
402         }
403         return 0;
404 }

406 int filter_comparison(int orig, int op)
407 {
408         if (orig == op)
409                 return orig;

411         orig = remove_unsigned_from_comparison(orig);
412         op = remove_unsigned_from_comparison(op);

414         switch (orig) {
415         case 0:
416                 return op;
417         case '<':
418                 switch (op) {
419                 case '<':
420                 case SPECIAL_LTE:
421                 case SPECIAL_NOTEQUAL:
422                         return '<';
423                 }
424                 return 0;
425         case SPECIAL_LTE:
426                 switch (op) {
427                 case '<':
428                 case SPECIAL_LTE:
429                 case SPECIAL_EQUAL:
430                         return op;
431                 case SPECIAL_NOTEQUAL:
432                         return '<';
433                 }
434                 return 0;
435         case SPECIAL_EQUAL:
436                 switch (op) {
437                 case SPECIAL_LTE:
438                 case SPECIAL_EQUAL:
439                 case SPECIAL_GTE:
440                 case SPECIAL_UNSIGNED_LTE:
441                 case SPECIAL_UNSIGNED_GTE:
442                         return SPECIAL_EQUAL;
443                 }
444                 return 0;
445         case SPECIAL_NOTEQUAL:
446                 switch (op) {
447                 case '<':
448                 case SPECIAL_LTE:
449                         return '<';
450                 case SPECIAL_UNSIGNED_LT:
451                 case SPECIAL_UNSIGNED_LTE:
452                         return SPECIAL_UNSIGNED_LT;
453                 case SPECIAL_NOTEQUAL:
454                         return op;
455                 case '>':
456                 case SPECIAL_GTE:
```



```
457                         return '>';
458                 case SPECIAL_UNSIGNED_GT:
459                 case SPECIAL_UNSIGNED_GTE:
460                         return SPECIAL_UNSIGNED_GT;
461                 }
462                 return 0;
463         case SPECIAL_GTE:
464                 switch (op) {
465                 case SPECIAL_LTE:
466                         return SPECIAL_EQUAL;
467                 case '>':
468                 case SPECIAL_GTE:
469                 case SPECIAL_EQUAL:
470                         return op;
471                 case SPECIAL_NOTEQUAL:
472                         return '>';
473                 }
474                 return 0;
475         case '>':
476                 switch (op) {
477                 case '>':
478                 case SPECIAL_GTE:
479                 case SPECIAL_NOTEQUAL:
480                         return '>';
481                 }
482                 return 0;
483         case SPECIAL_UNSIGNED_LT:
484                 switch (op) {
485                 case SPECIAL_UNSIGNED_LT:
486                 case SPECIAL_UNSIGNED_LTE:
487                 case SPECIAL_NOTEQUAL:
488                         return SPECIAL_UNSIGNED_LT;
489                 }
490                 return 0;
491         case SPECIAL_UNSIGNED_LTE:
492                 switch (op) {
493                 case SPECIAL_UNSIGNED_LT:
494                 case SPECIAL_UNSIGNED_LTE:
495                 case SPECIAL_EQUAL:
496                         return op;
497                 case SPECIAL_NOTEQUAL:
498                         return SPECIAL_UNSIGNED_LT;
499                 case SPECIAL_UNSIGNED_GTE:
500                         return SPECIAL_EQUAL;
501                 }
502                 return 0;
503         case SPECIAL_UNSIGNED_GTE:
504                 switch (op) {
505                 case SPECIAL_UNSIGNED_LTE:
506                         return SPECIAL_EQUAL;
507                 case SPECIAL_NOTEQUAL:
508                         return SPECIAL_UNSIGNED_GT;
509                 case SPECIAL_EQUAL:
510                 case SPECIAL_UNSIGNED_GTE:
511                 case SPECIAL_UNSIGNED_GT:
512                         return op;
513                 }
514                 return 0;
515         case SPECIAL_UNSIGNED_GT:
516                 switch (op) {
517                 case SPECIAL_UNSIGNED_GT:
518                 case SPECIAL_UNSIGNED_GTE:
519                 case SPECIAL_NOTEQUAL:
520                         return SPECIAL_UNSIGNED_GT;
521                 }
522                 return 0;
```

```
523         }
524         return 0;
525 }

527 static void pre_merge_hook(struct sm_state *sm)
528 {
529         struct compare_data *data = sm->state->data;
530         int other;

532         if (!data)
533                 return;
534         other = get_comparison(data->left, data->right);
535         if (!other)
536                 return;

538         set_state(compare_id, sm->name, NULL,
539                   alloc_compare_state(data->left, data->left_var, data->left_vsl
540                                       other,
541                                       data->right, data->right_var, data->right_
542 }

544 struct smatch_state *merge_compare_states(struct smatch_state *s1, struct smatch
545 {
546         struct compare_data *data = s1->data;
547         int op;

549         op = merge_comparisons(state_to_comparison(s1), state_to_comparison(s2))
550         if (op)
551                 return alloc_compare_state(
552                                 data->left, data->left_var, data->left_vsl,
553                                 op,
554                                 data->right, data->right_var, data->right_vsl);
555         return &undefined;
556 }

558 static struct smatch_state *alloc_link_state(struct string_list *links)
559 {
560         struct smatch_state *state;
561         static char buf[256];
562         char *tmp;
563         int i;

565         state = __alloc_smatch_state(0);

567         i = 0;
568         FOR_EACH_PTR(links, tmp) {
569                 if (!i++) {
570                         snprintf(buf, sizeof(buf), "%s", tmp);
571                 } else {
572                         append(buf, ", ", sizeof(buf));
573                         append(buf, tmp, sizeof(buf));
574                 }
575         } END_FOR_EACH_PTR(tmp);

577         state->name = alloc_sname(buf);
578         state->data = links;
579         return state;
580 }

582 static void save_start_states(struct statement *stmt)
583 {
584         struct symbol *param;
585         char orig[64];
586         char state_name[128];
587         struct smatch_state *state;
588         struct string_list *links;
```



```
589         char *link;

591         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, param) {
592                 struct var_sym_list *left_vsl = NULL;
593                 struct var_sym_list *right_vsl = NULL;

595                 if (!param->ident)
596                         continue;
597                 snprintf(orig, sizeof(orig), "%s orig", param->ident->name);
598                 snprintf(state_name, sizeof(state_name), "%s vs %s", param->iden
599                 add_var_sym(&left_vsl, param->ident->name, param);
600                 add_var_sym(&right_vsl, orig, param);
601                 state = alloc_compare_state(
602                                 NULL, param->ident->name, left_vsl,
603                                 SPECIAL_EQUAL,
604                                 NULL, alloc_sname(orig), right_vsl);
605                 set_state(compare_id, state_name, NULL, state);

607                 link = alloc_sname(state_name);
608                 links = NULL;
609                 insert_string(&links, link);
610                 state = alloc_link_state(links);
611                 set_state(link_id, param->ident->name, param, state);
612         } END_FOR_EACH_PTR(param);
613 }

615 static struct smatch_state *merge_links(struct smatch_state *s1, struct smatch_s
616 {
617         struct smatch_state *ret;
618         struct string_list *links;

620         links = combine_string_lists(s1->data, s2->data);
621         ret = alloc_link_state(links);
622         return ret;
623 }

625 static void save_link_var_sym(const char *var, struct symbol *sym, const char *l
626 {
627         struct smatch_state *old_state, *new_state;
628         struct string_list *links;
629         char *new;

631         old_state = get_state(link_id, var, sym);
632         if (old_state)
633                 links = clone_str_list(old_state->data);
634         else
635                 links = NULL;

637         new = alloc_sname(link);
638         insert_string(&links, new);

640         new_state = alloc_link_state(links);
641         set_state(link_id, var, sym, new_state);
642 }

644 static void match_inc(struct sm_state *sm)
645 {
646         struct string_list *links;
647         struct smatch_state *state, *new;
648         struct compare_data *data;
649         char *tmp;
650         int flip;
651         int op;

653         links = sm->state->data;
```

```
655         FOR_EACH_PTR(links, tmp) {
656                 state = get_state(compare_id, tmp, NULL);
657                 if (!state)
658                         continue;
659                 data = state->data;
660                 if (!data)
661                         continue;

663                 flip = 0;
664                 if (strncmp(sm->name, tmp, strlen(sm->name)) != 0 ||
665                     tmp[strlen(sm->name)] != ' ')
666                         flip = 1;

668                 op = state_to_comparison(state);

670                 switch (flip ? flip_comparison(op) : op) {
671                 case SPECIAL_EQUAL:
672                 case SPECIAL_GTE:
673                 case SPECIAL_UNSIGNED_GTE:
674                 case '>':
675                 case SPECIAL_UNSIGNED_GT:
676                         new = alloc_compare_state(
677                                         data->left, data->left_var, data->left_v
678                                         flip ? '<' : '>',
679                                         data->right, data->right_var, data->righ
680                         set_state(compare_id, tmp, NULL, new);
681                         break;
682                 case '<':
683                 case SPECIAL_UNSIGNED_LT:
684                         new = alloc_compare_state(
685                                         data->left, data->left_var, data->left_v
686                                         flip ? SPECIAL_GTE : SPECIAL_LTE,
687                                         data->right, data->right_var, data->righ
688                         set_state(compare_id, tmp, NULL, new);
689                         break;
690                 default:
691                         set_state(compare_id, tmp, NULL, &undefined);
692                 }
693         } END_FOR_EACH_PTR(tmp);
694 }

696 static void match_dec(struct sm_state *sm)
697 {
698         struct string_list *links;
699         struct smatch_state *state;
700         char *tmp;

702         links = sm->state->data;

704         FOR_EACH_PTR(links, tmp) {
705                 state = get_state(compare_id, tmp, NULL);

707                 switch (state_to_comparison(state)) {
708                 case SPECIAL_EQUAL:
709                 case SPECIAL_LTE:
710                 case SPECIAL_UNSIGNED_LTE:
711                 case '<':
712                 case SPECIAL_UNSIGNED_LT: {
713                         struct compare_data *data = state->data;
714                         struct smatch_state *new;

716                         new = alloc_compare_state(
717                                         data->left, data->left_var, data->left_v
718                                         '<',
719                                         data->right, data->right_var, data->righ
720                         set_state(compare_id, tmp, NULL, new);
```



```
721                         break;
722                         }
723                 default:
724                         set_state(compare_id, tmp, NULL, &undefined);
725                 }
726         } END_FOR_EACH_PTR(tmp);
727 }

729 static void match_inc_dec(struct sm_state *sm, struct expression *mod_expr)
730 {
731         /*
732          * if (foo > bar) then ++foo is also > bar.
733          */
734         if (!mod_expr)
735                 return;
736         if (mod_expr->type != EXPR_PREOP && mod_expr->type != EXPR_POSTOP)
737                 return;

739         if (mod_expr->op == SPECIAL_INCREMENT)
740                 match_inc(sm);
741         else if (mod_expr->op == SPECIAL_DECREMENT)
742                 match_dec(sm);
743 }

745 static int is_self_assign(struct expression *expr)
746 {
747         if (!expr || expr->type != EXPR_ASSIGNMENT || expr->op != '=')
748                 return 0;
749         return expr_equiv(expr->left, expr->right);
750 }

752 static void match_modify(struct sm_state *sm, struct expression *mod_expr)
753 {
754         struct string_list *links;
755         char *tmp;

757         if (mod_expr && is_self_assign(mod_expr))
758                 return;

760         /* handled by match_inc_dec() */
761         if (mod_expr &&
762             ((mod_expr->type == EXPR_PREOP || mod_expr->type == EXPR_POSTOP) &&
763              (mod_expr->op == SPECIAL_INCREMENT || mod_expr->op == SPECIAL_DECRE
764                 return;

766         links = sm->state->data;

768         FOR_EACH_PTR(links, tmp) {
769                 set_state(compare_id, tmp, NULL, &undefined);
770         } END_FOR_EACH_PTR(tmp);
771         set_state(link_id, sm->name, sm->sym, &undefined);
772 }

774 static void match_preop(struct expression *expr)
775 {
776         struct expression *parent;
777         struct range_list *left, *right;
778         int op;

780         /*
781          * This is an important special case.  Say you have:
782          *
783          *      if (++j == limit)
784          *
785          * Assume that we know the range of limit is higher than the start
786          * value for "j".  Then the first thing that we process is the ++j.  We
```

```
787          * have not comparison states set up so it doesn't get caught by the
788          * modification hook.  But it does get caught by smatch_extra which sets
789          * j to unknown then we parse the "j == limit" and sets false to != but
790          * really we want false to be <.
791          *
792          * So what we do is we set j < limit here, then the match_modify catches
793          * it and we do a match_inc_dec().
794          *
795          */
797         if (expr->type != EXPR_PREOP ||
798             (expr->op != SPECIAL_INCREMENT && expr->op != SPECIAL_DECREMENT))
799                 return;
801         parent = expr_get_parent_expr(expr);
802         if (!parent)
803                 return;
804         if (parent->type != EXPR_COMPARE || parent->op != SPECIAL_EQUAL)
805                 return;
806         if (parent->left != expr)
807                 return;
809         if (!get_implied_rl(expr->unop, &left) ||
810            !get_implied_rl(parent->right, &right))
811                 return;
813         op = rl_comparison(left, right);
814         if (!op)
815                 return;
817         add_comparison(expr->unop, op, parent->right);
818 }
820 static char *chunk_to_var_sym(struct expression *expr, struct symbol **sym)
821 {
822         expr = strip_expr(expr);
823         if (!expr)
824                 return NULL;
825         if (sym)
826                 *sym = NULL;
828         if (expr->type == EXPR_PREOP &&
829             (expr->op == SPECIAL_INCREMENT ||
830              expr->op == SPECIAL_DECREMENT))
831                 expr = strip_expr(expr->unop);
833         if (expr->type == EXPR_CALL) {
834                 char buf[64];
836                 snprintf(buf, sizeof(buf), "return %p", expr);
837                 return alloc_string(buf);
838         }
840         return expr_to_chunk_sym_vsl(expr, sym, NULL);
841 }
843 static char *chunk_to_var(struct expression *expr)
844 {
845         return chunk_to_var_sym(expr, NULL);
846 }
848 static struct smatch_state *get_state_chunk(int owner, struct expression *expr)
849 {
850         char *name;
851         struct symbol *sym;
852         struct smatch_state *ret;
```



```
854         name = chunk_to_var_sym(expr, &sym);
855         if (!name)
856                 return NULL;
858         ret = get_state(owner, name, sym);
859         free_string(name);
860         return ret;
861 }
863 static void save_link(struct expression *expr, char *link)
864 {
865         char *var;
866         struct symbol *sym;
868         expr = strip_expr(expr);
869         if (expr->type == EXPR_BINOP) {
870                 char *chunk;
872                 chunk = chunk_to_var(expr);
873                 if (!chunk)
874                         return;
876                 save_link(expr->left, link);
877                 save_link(expr->right, link);
878                 save_link_var_sym(chunk, NULL, link);
879                 return;
880         }
882         var = chunk_to_var_sym(expr, &sym);
883         if (!var)
884                 return;
886         save_link_var_sym(var, sym, link);
887         free_string(var);
888 }
890 static int get_orig_comparison(struct stree *pre_stree, const char *left, const
891 {
892         struct smatch_state *state;
893         struct compare_data *data;
894         int flip = 0;
895         char state_name[256];
897         if (strcmp(left, right) > 0) {
898                 const char *tmp = right;
900                 flip = 1;
901                 right = left;
902                 left = tmp;
903         }
905         snprintf(state_name, sizeof(state_name), "%s vs %s", left, right);
906         state = get_state_stree(pre_stree, compare_id, state_name, NULL);
907         if (!state || !state->data)
908                 return 0;
909         data = state->data;
910         if (flip)
911                 return flip_comparison(data->comparison);
912         return data->comparison;
914 }
916 static int have_common_var_sym(struct var_sym_list *left_vsl, struct var_sym_lis
917 {
918         struct var_sym *tmp;
```

```
920         FOR_EACH_PTR(left_vsl, tmp) {
921                 if (in_var_sym_list(right_vsl, tmp->var, tmp->sym))
922                         return 1;
923         } END_FOR_EACH_PTR(tmp);
924
925         return 0;
926 }
927
928 /*
929  * The idea here is that we take a comparison "a < b" and then we look at all
930  * the things which "b" is compared against "b < c" and we say that that implies
931  * a relationship "a < c".
932  *
933  * The names here about because the comparisons are organized like this
934  * "a < b < c".
935  *
936  */
937 static void update_tf_links(struct stree *pre_stree,
938                             struct expression *left_expr,
939                             const char *left_var, struct var_sym_list *left_vsl,
940                             int left_comparison, int left_false_comparison,
941                             const char *mid_var, struct var_sym_list *mid_vsl,
942                             struct string_list *links)
943 {
944         struct smatch_state *state;
945         struct smatch_state *true_state, *false_state;
946         struct compare_data *data;
947         struct expression *right_expr;
948         const char *right_var;
949         struct var_sym_list *right_vsl;
950         int orig_comparison;
951         int right_comparison;
952         int true_comparison;
953         int false_comparison;
954         char *tmp;
955         char state_name[256];
956         struct var_sym *vs;
957
958         FOR_EACH_PTR(links, tmp) {
959                 state = get_state_stree(pre_stree, compare_id, tmp, NULL);
960                 if (!state || !state->data)
961                         continue;
962                 data = state->data;
963                 right_comparison = data->comparison;
964                 right_expr = data->right;
965                 right_var = data->right_var;
966                 right_vsl = data->right_vsl;
967                 if (strcmp(mid_var, right_var) == 0) {
968                         right_expr = data->left;
969                         right_var = data->left_var;
970                         right_vsl = data->left_vsl;
971                         right_comparison = flip_comparison(right_comparison);
972                 }
973                 if (have_common_var_sym(left_vsl, right_vsl))
974                         continue;
975
976                 orig_comparison = get_orig_comparison(pre_stree, left_var, right
977
978                 true_comparison = combine_comparisons(left_comparison, right_com
979                 false_comparison = combine_comparisons(left_false_comparison, ri
980
981                 true_comparison = filter_comparison(orig_comparison, true_compar
982                 false_comparison = filter_comparison(orig_comparison, false_comp
983
984                 if (strcmp(left_var, right_var) > 0) {
```



```
985                         struct expression *tmp_expr = left_expr;
986                         const char *tmp_var = left_var;
987                         struct var_sym_list *tmp_vsl = left_vsl;
988
989                         left_expr = right_expr;
990                         left_var = right_var;
991                         left_vsl = right_vsl;
992                         right_expr = tmp_expr;
993                         right_var = tmp_var;
994                         right_vsl = tmp_vsl;
995                         true_comparison = flip_comparison(true_comparison);
996                         false_comparison = flip_comparison(false_comparison);
997                 }
998
999                 if (!true_comparison && !false_comparison)
1000                        continue;
1001
1002                if (true_comparison)
1003                        true_state = alloc_compare_state(
1004                                        left_expr, left_var, left_vsl,
1005                                        true_comparison,
1006                                        right_expr, right_var, right_vsl);
1007                else
1008                        true_state = NULL;
1009                if (false_comparison)
1010                        false_state = alloc_compare_state(
1011                                        left_expr, left_var, left_vsl,
1012                                        false_comparison,
1013                                        right_expr, right_var, right_vsl);
1014                else
1015                        false_state = NULL;
1016
1017                snprintf(state_name, sizeof(state_name), "%s vs %s", left_var, r
1018                set_true_false_states(compare_id, state_name, NULL, true_state,
1019                FOR_EACH_PTR(left_vsl, vs) {
1020                        save_link_var_sym(vs->var, vs->sym, state_name);
1021                } END_FOR_EACH_PTR(vs);
1022                FOR_EACH_PTR(right_vsl, vs) {
1023                        save_link_var_sym(vs->var, vs->sym, state_name);
1024                } END_FOR_EACH_PTR(vs);
1025                if (!vsl_to_sym(left_vsl))
1026                        save_link_var_sym(left_var, NULL, state_name);
1027                if (!vsl_to_sym(right_vsl))
1028                        save_link_var_sym(right_var, NULL, state_name);
1029        } END_FOR_EACH_PTR(tmp);
1030 }
1031
1032 static void update_tf_data(struct stree *pre_stree,
1033                 struct expression *left_expr,
1034                 const char *left_name, struct var_sym_list *left_vsl,
1035                 struct expression *right_expr,
1036                 const char *right_name, struct var_sym_list *right_vsl,
1037                 int true_comparison, int false_comparison)
1038 {
1039         struct smatch_state *state;
1040
1041         state = get_state_stree(pre_stree, link_id, right_name, vsl_to_sym(right
1042         if (state)
1043                 update_tf_links(pre_stree, left_expr, left_name, left_vsl, true_
1044
1045         state = get_state_stree(pre_stree, link_id, left_name, vsl_to_sym(left_v
1046         if (state)
1047                 update_tf_links(pre_stree, right_expr, right_name, right_vsl, fl
1048 }
1049
1050 static void iter_modify(struct sm_state *sm, struct expression *mod_expr)
```

```
1051 {
1052         if (sm->state != &start ||
1053             !mod_expr ||
1054             (mod_expr->type != EXPR_PREOP && mod_expr->type != EXPR_POSTOP) ||
1055             mod_expr->op != SPECIAL_INCREMENT)
1056                 set_state(inc_dec_id, sm->name, sm->sym, &undefined);
1057         else
1058                 set_state(inc_dec_id, sm->name, sm->sym, &incremented);
1059 }

1061 static void handle_for_loops(struct expression *expr, char *state_name, struct s
1062 {
1063         sval_t sval;
1064         char *iter_name, *cap_name;
1065         struct symbol *iter_sym, *cap_sym;
1066         struct compare_data *data;

1068         if (expr->op != '<' && expr->op != SPECIAL_UNSIGNED_LT)
1069                 return;

1071         if (!__cur_stmt || !__prev_stmt)
1072                 return;
1073         if (__cur_stmt->type != STMT_ITERATOR)
1074                 return;
1075         if (__cur_stmt->iterator_pre_condition != expr)
1076                 return;

1078         /* literals are handled in smatch_extra.c */
1079         if (get_value(expr->right, &sval))
1080                 return;

1082         /* First time checking the condition */
1083         if (__prev_stmt == __cur_stmt->iterator_pre_statement) {
1084                 if (!get_implied_value(expr->left, &sval) ||
1085                     sval.value != 0)
1086                         return;

1088                 iter_name = expr_to_var_sym(expr->left, &iter_sym);
1089                 cap_name = expr_to_var_sym(expr->right, &cap_sym);
1090                 if (!iter_name || !cap_name || !iter_sym || !cap_sym) {
1091                         free_string(iter_name);
1092                         free_string(cap_name);
1093                         return;
1094                 }

1096                 set_state(inc_dec_id, iter_name, iter_sym, &start);
1097                 store_link(inc_dec_link_id, cap_name, cap_sym, iter_name, iter_s

1099                 free_string(iter_name);
1100                 free_string(cap_name);
1101                 return;
1102         }

1104         /* Second time checking the condtion */
1105         if (__prev_stmt != __cur_stmt->iterator_post_statement)
1106                 return;

1108         if (get_state_chunk(inc_dec_id, expr->left) != &incremented)
1109                 return;

1111         data = false_state->data;
1112         false_state = alloc_compare_state(
1113                         data->left, data->left_var, data->left_vsl,
1114                         SPECIAL_EQUAL,
1115                         data->right, data->right_var, data->right_vsl);
```



```
1117         set_true_false_states(compare_id, state_name, NULL, NULL, false_state);
1118 }

1120 static int is_plus_one(struct expression *expr)
1121 {
1122         sval_t sval;

1124         if (expr->type != EXPR_BINOP || expr->op != '+')
1125                 return 0;
1126         if (!get_implied_value(expr->right, &sval) || sval.value != 1)
1127                 return 0;
1128         return 1;
1129 }

1131 static int is_minus_one(struct expression *expr)
1132 {
1133         sval_t sval;

1135         if (expr->type != EXPR_BINOP || expr->op != '-')
1136                 return 0;
1137         if (!get_implied_value(expr->right, &sval) || sval.value != 1)
1138                 return 0;
1139         return 1;
1140 }

1142 static void move_plus_to_minus_helper(struct expression **left_p, struct express
1143 {
1144         struct expression *left = *left_p;
1145         struct expression *right = *right_p;

1147         /*
1148          * These two are basically equivalent: "foo + 1 != bar" and
1149          * "foo != bar - 1".  There are issues with signedness and integer
1150          * overflows.  There are also issues with type as well.  But let's
1151          * pretend we can ignore all that stuff for now.
1152          *
1153          */

1155         if (!is_plus_one(left))
1156                 return;

1158         *left_p = left->left;
1159         *right_p = binop_expression(right, '-', left->right);
1160 }

1162 static void move_plus_to_minus(struct expression **left_p, struct expression **r
1163 {
1164         if (is_plus_one(*left_p) && is_plus_one(*right_p))
1165                 return;

1167         move_plus_to_minus_helper(left_p, right_p);
1168         move_plus_to_minus_helper(right_p, left_p);
1169 }

1171 static void handle_comparison(struct expression *left_expr, int op, struct expre
1172 {
1173         char *left = NULL;
1174         char *right = NULL;
1175         struct symbol *left_sym, *right_sym;
1176         struct var_sym_list *left_vsl = NULL;
1177         struct var_sym_list *right_vsl = NULL;
1178         int false_op;
1179         int orig_comparison;
1180         struct smatch_state *true_state, *false_state;
1181         static char state_name[256];
1182         struct stree *pre_stree;
```

```
1183         sval_t sval;
1185         if (!left_expr || !right_expr)
1186                 return;
1188         left_expr = strip_parens(left_expr);
1189         right_expr = strip_parens(right_expr);
1191         while (left_expr->type == EXPR_ASSIGNMENT)
1192                 left_expr = strip_parens(left_expr->left);
1193         while (right_expr->type == EXPR_ASSIGNMENT)
1194                 right_expr = strip_parens(right_expr->left);
1196         false_op = negate_comparison(op);
1198         move_plus_to_minus(&left_expr, &right_expr);
1200         if (op == SPECIAL_UNSIGNED_LT &&
1201             get_implied_value(left_expr, &sval) &&
1202             sval.value == 0)
1203                 false_op = SPECIAL_EQUAL;
1205         if (op == SPECIAL_UNSIGNED_GT &&
1206             get_implied_value(right_expr, &sval) &&
1207             sval.value == 0)
1208                 false_op = SPECIAL_EQUAL;
1210         left = chunk_to_var_sym(left_expr, &left_sym);
1211         if (!left)
1212                 goto free;
1213         if (left_sym)
1214                 add_var_sym(&left_vsl, left, left_sym);
1215         else
1216                 left_vsl = expr_to_vsl(left_expr);
1217         right = chunk_to_var_sym(right_expr, &right_sym);
1218         if (!right)
1219                 goto free;
1220         if (right_sym)
1221                 add_var_sym(&right_vsl, right, right_sym);
1222         else
1223                 right_vsl = expr_to_vsl(right_expr);
1225         if (strcmp(left, right) > 0) {
1226                 char *tmp_name = left;
1227                 struct var_sym_list *tmp_vsl = left_vsl;
1228                 struct expression *tmp_expr = left_expr;
1230                 left = right;
1231                 left_vsl = right_vsl;
1232                 left_expr = right_expr;
1233                 right = tmp_name;
1234                 right_vsl = tmp_vsl;
1235                 right_expr = tmp_expr;
1236                 op = flip_comparison(op);
1237                 false_op = flip_comparison(false_op);
1238         }
1240         orig_comparison = get_comparison(left_expr, right_expr);
1241         op = filter_comparison(orig_comparison, op);
1242         false_op = filter_comparison(orig_comparison, false_op);
1244         snprintf(state_name, sizeof(state_name), "%s vs %s", left, right);
1245         true_state = alloc_compare_state(
1246                         left_expr, left, left_vsl,
1247                         op,
1248                         right_expr, right, right_vsl);
```



```
1249         false_state = alloc_compare_state(
1250                         left_expr, left, left_vsl,
1251                         false_op,
1252                         right_expr, right, right_vsl);
1254         pre_stree = clone_stree(__get_cur_stree());
1255         update_tf_data(pre_stree, left_expr, left, left_vsl, right_expr, right,
1256         free_stree(&pre_stree);
1258         set_true_false_states(compare_id, state_name, NULL, true_state, false_st
1259         __compare_param_limit_hook(left_expr, right_expr, state_name, true_state
1260         save_link(left_expr, state_name);
1261         save_link(right_expr, state_name);
1263         if (_false_state)
1264                 *_false_state = false_state;
1265         if (_state_name)
1266                 *_state_name = state_name;
1267 free:
1268         free_string(left);
1269         free_string(right);
1270 }
1272 void __comparison_match_condition(struct expression *expr)
1273 {
1274         struct expression *left, *right, *new_left, *new_right, *tmp;
1275         struct smatch_state *false_state = NULL;
1276         char *state_name = NULL;
1277         int redo, count;
1279         if (expr->type != EXPR_COMPARE)
1280                 return;
1282         handle_comparison(expr->left, expr->op, expr->right, &state_name, &false
1283         if (false_state && state_name)
1284                 handle_for_loops(expr, state_name, false_state);
1286         left = strip_parens(expr->left);
1287         right = strip_parens(expr->right);
1289         if (left->type == EXPR_BINOP && left->op == '+') {
1290                 new_left = left->left;
1291                 new_right = binop_expression(right, '-', left->right);
1292                 handle_comparison(new_left, expr->op, new_right, NULL, NULL);
1294                 new_left = left->right;
1295                 new_right = binop_expression(right, '-', left->left);
1296                 handle_comparison(new_left, expr->op, new_right, NULL, NULL);
1297         }
1300         redo = 0;
1301         left = strip_parens(expr->left);
1302         right = strip_parens(expr->right);
1303         if (get_last_expr_from_expression_stmt(expr->left)) {
1304                 left = get_last_expr_from_expression_stmt(expr->left);
1305                 redo = 1;
1306         }
1307         if (get_last_expr_from_expression_stmt(expr->right)) {
1308                 right = get_last_expr_from_expression_stmt(expr->right);
1309                 redo = 1;
1310         }
1312         if (!redo)
1313                 return;
```

```
1315         count = 0;
1316         while ((tmp = get_assigned_expr(left))) {
1317                 if (count++ > 3)
1318                         break;
1319                 left = strip_expr(tmp);
1320         }
1321         count = 0;
1322         while ((tmp = get_assigned_expr(right))) {
1323                 if (count++ > 3)
1324                         break;
1325                 right = strip_expr(tmp);
1326         }

1328         handle_comparison(left, expr->op, right, NULL, NULL);
1329 }

1331 static void add_comparison_var_sym(
1332                 struct expression *left_expr,
1333                 const char *left_name, struct var_sym_list *left_vsl,
1334                 int comparison,
1335                 struct expression *right_expr,
1336                 const char *right_name, struct var_sym_list *right_vsl)
1337 {
1338         struct smatch_state *state;
1339         struct var_sym *vs;
1340         char state_name[256];

1342         if (strcmp(left_name, right_name) > 0) {
1343                 struct expression *tmp_expr = left_expr;
1344                 const char *tmp_name = left_name;
1345                 struct var_sym_list *tmp_vsl = left_vsl;

1347                 left_expr = right_expr;
1348                 left_name = right_name;
1349                 left_vsl = right_vsl;
1350                 right_expr = tmp_expr;
1351                 right_name = tmp_name;
1352                 right_vsl = tmp_vsl;
1353                 comparison = flip_comparison(comparison);
1354         }
1355         snprintf(state_name, sizeof(state_name), "%s vs %s", left_name, right_na
1356         state = alloc_compare_state(
1357                         left_expr, left_name, left_vsl,
1358                         comparison,
1359                         right_expr, right_name, right_vsl);

1361         set_state(compare_id, state_name, NULL, state);

1363         FOR_EACH_PTR(left_vsl, vs) {
1364                 save_link_var_sym(vs->var, vs->sym, state_name);
1365         } END_FOR_EACH_PTR(vs);
1366         FOR_EACH_PTR(right_vsl, vs) {
1367                 save_link_var_sym(vs->var, vs->sym, state_name);
1368         } END_FOR_EACH_PTR(vs);
1369 }

1371 static void add_comparison(struct expression *left, int comparison, struct expre
1372 {
1373         char *left_name = NULL;
1374         char *right_name = NULL;
1375         struct symbol *left_sym, *right_sym;
1376         struct var_sym_list *left_vsl, *right_vsl;
1377         struct smatch_state *state;
1378         char state_name[256];

1380         left_name = chunk_to_var_sym(left, &left_sym);
```



```
1381         if (!left_name)
1382                 goto free;
1383         left_vsl = expr_to_vsl(left);
1384         right_name = chunk_to_var_sym(right, &right_sym);
1385         if (!right_name)
1386                 goto free;
1387         right_vsl = expr_to_vsl(right);

1389         if (strcmp(left_name, right_name) > 0) {
1390                 struct expression *tmp_expr = left;
1391                 struct symbol *tmp_sym = left_sym;
1392                 char *tmp_name = left_name;
1393                 struct var_sym_list *tmp_vsl = left_vsl;

1395                 left = right;
1396                 left_name = right_name;
1397                 left_sym = right_sym;
1398                 left_vsl = right_vsl;
1399                 right = tmp_expr;
1400                 right_name = tmp_name;
1401                 right_sym = tmp_sym;
1402                 right_vsl = tmp_vsl;
1403                 comparison = flip_comparison(comparison);
1404         }
1405         snprintf(state_name, sizeof(state_name), "%s vs %s", left_name, right_na
1406         state = alloc_compare_state(
1407                         left, left_name, left_vsl,
1408                         comparison,
1409                         right, right_name, right_vsl);

1411         set_state(compare_id, state_name, NULL, state);
1412         save_link(left, state_name);
1413         save_link(right, state_name);

1415 free:
1416         free_string(left_name);
1417         free_string(right_name);
1418 }

1420 static void match_assign_add(struct expression *expr)
1421 {
1422         struct expression *right;
1423         struct expression *r_left, *r_right;
1424         sval_t left_tmp, right_tmp;

1426         right = strip_expr(expr->right);
1427         r_left = strip_expr(right->left);
1428         r_right = strip_expr(right->right);

1430         get_absolute_min(r_left, &left_tmp);
1431         get_absolute_min(r_right, &right_tmp);

1433         if (left_tmp.value > 0)
1434                 add_comparison(expr->left, '>', r_right);
1435         else if (left_tmp.value == 0)
1436                 add_comparison(expr->left, SPECIAL_GTE, r_right);

1438         if (right_tmp.value > 0)
1439                 add_comparison(expr->left, '>', r_left);
1440         else if (right_tmp.value == 0)
1441                 add_comparison(expr->left, SPECIAL_GTE, r_left);
1442 }

1444 static void match_assign_sub(struct expression *expr)
1445 {
1446         struct expression *right;
```

```
1447         struct expression *r_left, *r_right;
1448         int comparison;
1449         sval_t min;

1451         right = strip_expr(expr->right);
1452         r_left = strip_expr(right->left);
1453         r_right = strip_expr(right->right);

1455         if (get_absolute_min(r_right, &min) && sval_is_negative(min))
1456                 return;

1458         comparison = get_comparison(r_left, r_right);

1460         switch (comparison) {
1461         case '>':
1462         case SPECIAL_GTE:
1463                 if (implied_not_equal(r_right, 0))
1464                         add_comparison(expr->left, '>', r_left);
1465                 else
1466                         add_comparison(expr->left, SPECIAL_GTE, r_left);
1467                 return;
1468         }
1469 }

1471 static void match_assign_divide(struct expression *expr)
1472 {
1473         struct expression *right;
1474         struct expression *r_left, *r_right;
1475         sval_t min;

1477         right = strip_expr(expr->right);
1478         r_left = strip_expr(right->left);
1479         r_right = strip_expr(right->right);
1480         if (!get_implied_min(r_right, &min) || min.value <= 1)
1481                 return;

1483         add_comparison(expr->left, '<', r_left);
1484 }

1486 static void match_binop_assign(struct expression *expr)
1487 {
1488         struct expression *right;

1490         right = strip_expr(expr->right);
1491         if (right->op == '+')
1492                 match_assign_add(expr);
1493         if (right->op == '-')
1494                 match_assign_sub(expr);
1495         if (right->op == '/')
1496                 match_assign_divide(expr);
1497 }

1499 static void copy_comparisons(struct expression *left, struct expression *right)
1500 {
1501         struct string_list *links;
1502         struct smatch_state *state;
1503         struct compare_data *data;
1504         struct symbol *left_sym, *right_sym;
1505         char *left_var = NULL;
1506         char *right_var = NULL;
1507         struct var_sym_list *left_vsl;
1508         struct expression *expr;
1509         const char *var;
1510         struct var_sym_list *vsl;
1511         int comparison;
1512         char *tmp;
```



```
1514         left_var = chunk_to_var_sym(left, &left_sym);
1515         if (!left_var)
1516                 goto done;
1517         left_vsl = expr_to_vsl(left);
1518         right_var = chunk_to_var_sym(right, &right_sym);
1519         if (!right_var)
1520                 goto done;

1522         state = get_state(link_id, right_var, right_sym);
1523         if (!state)
1524                 return;
1525         links = state->data;

1527         FOR_EACH_PTR(links, tmp) {
1528                 state = get_state(compare_id, tmp, NULL);
1529                 if (!state || !state->data)
1530                         continue;
1531                 data = state->data;
1532                 comparison = data->comparison;
1533                 expr = data->right;
1534                 var = data->right_var;
1535                 vsl = data->right_vsl;
1536                 if (strcmp(var, right_var) == 0) {
1537                         expr = data->left;
1538                         var = data->left_var;
1539                         vsl = data->left_vsl;
1540                         comparison = flip_comparison(comparison);
1541                 }
1542                 /* n = copy_from_user(dest, src, n); leads to n <= n which is no
1543                 if (strcmp(left_var, var) == 0)
1544                         continue;
1545                 add_comparison_var_sym(left, left_var, left_vsl, comparison, exp
1546         } END_FOR_EACH_PTR(tmp);

1548 done:
1549         free_string(right_var);
1550 }

1552 static void match_assign(struct expression *expr)
1553 {
1554         struct expression *right;

1556         if (expr->op != '=')
1557                 return;
1558         if (__in_fake_assign || outside_of_function())
1559                 return;

1561         if (is_struct(expr->left))
1562                 return;

1564         if (is_self_assign(expr))
1565                 return;

1567         copy_comparisons(expr->left, expr->right);
1568         add_comparison(expr->left, SPECIAL_EQUAL, expr->right);

1570         right = strip_expr(expr->right);
1571         if (right->type == EXPR_BINOP)
1572                 match_binop_assign(expr);
1573 }

1575 int get_comparison_strings(const char *one, const char *two)
1576 {
1577         char buf[256];
1578         struct smatch_state *state;
```

```
1579         int invert = 0;
1580         int ret = 0;

1582         if (!one || !two)
1583                 return 0;

1585         if (strcmp(one, two) == 0)
1586                 return SPECIAL_EQUAL;

1588         if (strcmp(one, two) > 0) {
1589                 const char *tmp = one;

1591                 one = two;
1592                 two = tmp;
1593                 invert = 1;
1594         }

1596         snprintf(buf, sizeof(buf), "%s vs %s", one, two);
1597         state = get_state(compare_id, buf, NULL);
1598         if (state)
1599                 ret = state_to_comparison(state);

1601         if (invert)
1602                 ret = flip_comparison(ret);

1604         return ret;
1605 }

1607 int get_comparison(struct expression *a, struct expression *b)
1608 {
1609         char *one = NULL;
1610         char *two = NULL;
1611         int ret = 0;

1613         if (!a || !b)
1614                 return 0;

1616         a = strip_parens(a);
1617         b = strip_parens(b);

1619         move_plus_to_minus(&a, &b);

1621         one = chunk_to_var(a);
1622         if (!one)
1623                 goto free;
1624         two = chunk_to_var(b);
1625         if (!two)
1626                 goto free;

1628         ret = get_comparison_strings(one, two);
1629         if (ret)
1630                 goto free;

1632         if (is_plus_one(a) || is_minus_one(a)) {
1633                 free_string(one);
1634                 one = chunk_to_var(a->left);
1635                 ret = get_comparison_strings(one, two);
1636         } else if (is_plus_one(b) || is_minus_one(b)) {
1637                 free_string(two);
1638                 two = chunk_to_var(b->left);
1639                 ret = get_comparison_strings(one, two);
1640         }

1642         if (!ret)
1643                 goto free;
```



```
1645         if ((is_plus_one(a) || is_minus_one(b)) && ret == '<')
1646                 ret = SPECIAL_LTE;
1647         else if ((is_minus_one(a) || is_plus_one(b)) && ret == '>')
1648                 ret = SPECIAL_GTE;
1649         else
1650                 ret = 0;

1652 free:
1653         free_string(one);
1654         free_string(two);

1656         if (!ret)
1657                 return comparison_from_extra(a, b);
1658         return ret;
1659 }

1661 int possible_comparison(struct expression *a, int comparison, struct expression
1662 {
1663         char *one = NULL;
1664         char *two = NULL;
1665         int ret = 0;
1666         char buf[256];
1667         struct sm_state *sm;
1668         int saved;

1670         one = chunk_to_var(a);
1671         if (!one)
1672                 goto free;
1673         two = chunk_to_var(b);
1674         if (!two)
1675                 goto free;


1678         if (strcmp(one, two) == 0 && comparison == SPECIAL_EQUAL) {
1679                 ret = 1;
1680                 goto free;
1681         }

1683         if (strcmp(one, two) > 0) {
1684                 char *tmp = one;

1686                 one = two;
1687                 two = tmp;
1688                 comparison = flip_comparison(comparison);
1689         }

1691         snprintf(buf, sizeof(buf), "%s vs %s", one, two);
1692         sm = get_sm_state(compare_id, buf, NULL);
1693         if (!sm)
1694                 goto free;

1696         FOR_EACH_PTR(sm->possible, sm) {
1697                 if (!sm->state->data)
1698                         continue;
1699                 saved = ((struct compare_data *)sm->state->data)->comparison;
1700                 if (saved == comparison)
1701                         ret = 1;
1702                 if (comparison == SPECIAL_EQUAL &&
1703                     (saved == SPECIAL_LTE ||
1704                      saved == SPECIAL_GTE ||
1705                      saved == SPECIAL_UNSIGNED_LTE ||
1706                      saved == SPECIAL_UNSIGNED_GTE))
1707                         ret = 1;
1708                 if (ret == 1)
1709                         goto free;
1710         } END_FOR_EACH_PTR(sm);
```

```
1712         return ret;
1713 free:
1714         free_string(one);
1715         free_string(two);
1716         return ret;
1717 }

1719 struct state_list *get_all_comparisons(struct expression *expr)
1720 {
1721         struct smatch_state *state;
1722         struct string_list *links;
1723         struct state_list *ret = NULL;
1724         struct sm_state *sm;
1725         char *tmp;

1727         state = get_state_chunk(link_id, expr);
1728         if (!state)
1729                 return NULL;
1730         links = state->data;

1732         FOR_EACH_PTR(links, tmp) {
1733                 sm = get_sm_state(compare_id, tmp, NULL);
1734                 if (!sm)
1735                         continue;
1736                 // FIXME have to compare name with vsl
1737                 add_ptr_list(&ret, sm);
1738         } END_FOR_EACH_PTR(tmp);

1740         return ret;
1741 }

1743 struct state_list *get_all_possible_equal_comparisons(struct expression *expr)
1744 {
1745         struct smatch_state *state;
1746         struct string_list *links;
1747         struct state_list *ret = NULL;
1748         struct sm_state *sm;
1749         char *tmp;

1751         state = get_state_chunk(link_id, expr);
1752         if (!state)
1753                 return NULL;
1754         links = state->data;

1756         FOR_EACH_PTR(links, tmp) {
1757                 sm = get_sm_state(compare_id, tmp, NULL);
1758                 if (!sm)
1759                         continue;
1760                 if (!strchr(sm->state->name, '='))
1761                         continue;
1762                 if (strcmp(sm->state->name, "!=") == 0)
1763                         continue;
1764                 add_ptr_list(&ret, sm);
1765         } END_FOR_EACH_PTR(tmp);

1767         return ret;
1768 }

1770 struct state_list *get_all_possible_not_equal_comparisons(struct expression *exp
1771 {
1772         struct smatch_state *state;
1773         struct string_list *links;
1774         struct state_list *ret = NULL;
1775         struct sm_state *sm;
1776         struct sm_state *possible;
```



```
1777         char *link;

1779         return NULL;

1781         state = get_state_chunk(link_id, expr);
1782         if (!state)
1783                 return NULL;
1784         links = state->data;

1786         FOR_EACH_PTR(links, link) {
1787                 sm = get_sm_state(compare_id, link, NULL);
1788                 if (!sm)
1789                         continue;
1790                 FOR_EACH_PTR(sm->possible, possible) {
1791                         if (strcmp(possible->state->name, "!=") != 0)
1792                                 continue;
1793                         add_ptr_list(&ret, sm);
1794                         break;
1795                 } END_FOR_EACH_PTR(link);
1796         } END_FOR_EACH_PTR(link);

1798         return ret;
1799 }

1801 static void update_links_from_call(struct expression *left,
1802                                    int left_compare,
1803                                    struct expression *right)
1804 {
1805         struct string_list *links;
1806         struct smatch_state *state;
1807         struct compare_data *data;
1808         struct symbol *left_sym, *right_sym;
1809         char *left_var = NULL;
1810         char *right_var = NULL;
1811         struct var_sym_list *left_vsl;
1812         struct expression *expr;
1813         const char *var;
1814         struct var_sym_list *vsl;
1815         int comparison;
1816         char *tmp;

1818         left_var = chunk_to_var_sym(left, &left_sym);
1819         if (!left_var)
1820                 goto done;
1821         left_vsl = expr_to_vsl(left);
1822         right_var = chunk_to_var_sym(right, &right_sym);
1823         if (!right_var)
1824                 goto done;

1826         state = get_state(link_id, right_var, right_sym);
1827         if (!state)
1828                 return;
1829         links = state->data;

1831         FOR_EACH_PTR(links, tmp) {
1832                 state = get_state(compare_id, tmp, NULL);
1833                 if (!state || !state->data)
1834                         continue;
1835                 data = state->data;
1836                 comparison = data->comparison;
1837                 expr = data->right;
1838                 var = data->right_var;
1839                 vsl = data->right_vsl;
1840                 if (strcmp(var, right_var) == 0) {
1841                         expr = data->left;
1842                         var = data->left_var;
```

```
1843                        vsl = data->left_vsl;
1844                        comparison = flip_comparison(comparison);
1845                }
1846                comparison = combine_comparisons(left_compare, comparison);
1847                if (!comparison)
1848                        continue;
1849                add_comparison_var_sym(left, left_var, left_vsl, comparison, exp
1850        } END_FOR_EACH_PTR(tmp);

1852 done:
1853        free_string(right_var);
1854 }

1856 void __add_return_comparison(struct expression *call, const char *range)
1857 {
1858        struct expression *arg;
1859        int comparison;
1860        char buf[4];

1862        if (!str_to_comparison_arg(range, call, &comparison, &arg))
1863                return;
1864        snprintf(buf, sizeof(buf), "%s", show_special(comparison));
1865        update_links_from_call(call, comparison, arg);
1866        add_comparison(call, comparison, arg);
1867 }

1869 void __add_comparison_info(struct expression *expr, struct expression *call, con
1870 {
1871        copy_comparisons(expr, call);
1872 }

1874 static char *get_mask_comparison(struct expression *expr, int ignore)
1875 {
1876        struct expression *tmp, *right;
1877        int count, param;
1878        char buf[256];

1880        /* The return value for "return foo & param;" is <= param */

1882        count = 0;
1883        while ((tmp = get_assigned_expr(expr))) {
1884                expr = strip_expr(tmp);
1885                if (count++ > 4)
1886                        break;
1887        }

1889        if (expr->type != EXPR_BINOP || expr->op != '&')
1890                return NULL;

1892        right = strip_expr(expr->right);
1893        param = get_param_num(right);
1894        if (param < 0 || param == ignore)
1895                return NULL;

1897        snprintf(buf, sizeof(buf), "[<=$%d]", param);
1898        return alloc_sname(buf);
1899 }

1901 static char *range_comparison_to_param_helper(struct expression *expr, char star
1902 {
1903        struct symbol *param;
1904        char *var = NULL;
1905        char buf[256];
1906        char *ret_str = NULL;
1907        int compare;
1908        int i;
```



```
1910        if (!expr)
1911                return NULL;

1913        var = chunk_to_var(expr);
1914        if (!var)
1915                goto try_mask;

1917        i = -1;
1918        FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, param) {
1919                i++;
1920                if (i == ignore)
1921                        continue;
1922                if (!param->ident)
1923                        continue;
1924                snprintf(buf, sizeof(buf), "%s orig", param->ident->name);
1925                compare = get_comparison_strings(var, buf);
1926                if (!compare)
1927                        continue;
1928                if (show_special(compare)[0] != starts_with)
1929                        continue;
1930                snprintf(buf, sizeof(buf), "[%s$%d]", show_special(compare), i);
1931                ret_str = alloc_sname(buf);
1932                break;
1933        } END_FOR_EACH_PTR(param);

1935        free_string(var);
1936        if (!ret_str)
1937                goto try_mask;

1939        return ret_str;

1941 try_mask:
1942        if (starts_with == '<')
1943                ret_str = get_mask_comparison(expr, ignore);
1944        return ret_str;
1945 }

1947 char *name_sym_to_param_comparison(const char *name, struct symbol *sym)
1948 {
1949        struct symbol *param;
1950        char buf[256];
1951        int compare;
1952        int i;

1954        i = -1;
1955        FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, param) {
1956                i++;
1957                if (!param->ident)
1958                        continue;
1959                snprintf(buf, sizeof(buf), "%s orig", param->ident->name);
1960                compare = get_comparison_strings(name, buf);
1961                if (!compare)
1962                        continue;
1963                snprintf(buf, sizeof(buf), "[%s$%d]", show_special(compare), i);
1964                return alloc_sname(buf);
1965        } END_FOR_EACH_PTR(param);

1967        return NULL;
1968 }

1970 char *expr_equal_to_param(struct expression *expr, int ignore)
1971 {
1972        return range_comparison_to_param_helper(expr, '=', ignore);
1973 }
```

```
1975 char *expr_lte_to_param(struct expression *expr, int ignore)
1976 {
1977         return range_comparison_to_param_helper(expr, '<', ignore);
1978 }

1980 char *expr_param_comparison(struct expression *expr, int ignore)
1981 {
1982         struct symbol *param;
1983         char *var = NULL;
1984         char buf[256];
1985         char *ret_str = NULL;
1986         int compare;
1987         int i;

1989         var = chunk_to_var(expr);
1990         if (!var)
1991                 goto free;

1993         i = -1;
1994         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, param) {
1995                 i++;
1996                 if (i == ignore)
1997                         continue;
1998                 if (!param->ident)
1999                         continue;
2000                 snprintf(buf, sizeof(buf), "%s orig", param->ident->name);
2001                 compare = get_comparison_strings(var, buf);
2002                 if (!compare)
2003                         continue;
2004                 snprintf(buf, sizeof(buf), "[%s$%d]", show_special(compare), i);
2005                 ret_str = alloc_sname(buf);
2006                 break;
2007         } END_FOR_EACH_PTR(param);

2009 free:
2010         free_string(var);
2011         return ret_str;
2012 }

2014 char *get_printed_param_name(struct expression *call, const char *param_name, st
2015 {
2016         struct expression *arg;
2017         char *name;
2018         struct symbol *sym;
2019         static char buf[256];
2020         int len;
2021         int i;

2023         i = -1;
2024         FOR_EACH_PTR(call->args, arg) {
2025                 i++;

2027                 name = expr_to_var_sym(arg, &sym);
2028                 if (!name || !sym)
2029                         continue;
2030                 if (sym != param_sym)
2031                         continue;

2033                 len = strlen(name);
2034                 if (strncmp(name, param_name, len) != 0)
2035                         continue;
2036                 if (param_name[len] == '\0') {
2037                         snprintf(buf, sizeof(buf), "$%d", i);
2038                         return buf;
2039                 }
2040                 if (param_name[len] != '-')
```



```
2041                         continue;
2042                 snprintf(buf, sizeof(buf), "$%d%s", i, param_name + len);
2043                 return buf;
2044         } END_FOR_EACH_PTR(arg);

2046         return NULL;
2047 }

2049 static void match_call_info(struct expression *expr)
2050 {
2051         struct expression *arg;
2052         struct smatch_state *state;
2053         struct sm_state *sm;
2054         struct compare_data *data;
2055         int comparison;
2056         struct string_list *links;
2057         char *arg_name;
2058         const char *right_name;
2059         char *link;
2060         char info_buf[256];
2061         int i;

2063         i = -1;
2064         FOR_EACH_PTR(expr->args, arg) {
2065                 i++;

2067                 state = get_state_chunk(link_id, arg);
2068                 if (!state)
2069                         continue;

2071                 links = state->data;
2072                 FOR_EACH_PTR(links, link) {
2073                         struct var_sym_list *right_vsl;
2074                         struct var_sym *right_vs;


2077                         if (strstr(link, " orig"))
2078                                 continue;
2079                         sm = get_sm_state(compare_id, link, NULL);
2080                         if (!sm)
2081                                 continue;
2082                         data = sm->state->data;
2083                         if (!data || !data->comparison)
2084                                 continue;
2085                         arg_name = expr_to_var(arg);
2086                         if (!arg_name)
2087                                 continue;

2089                         right_vsl = NULL;
2090                         if (strcmp(data->left_var, arg_name) == 0) {
2091                                 comparison = data->comparison;
2092                                 right_name = data->right_var;
2093                                 right_vsl = data->right_vsl;
2094                         } else if (strcmp(data->right_var, arg_name) == 0) {
2095                                 comparison = flip_comparison(data->comparison);
2096                                 right_name = data->left_var;
2097                                 right_vsl = data->left_vsl;
2098                         }
2099                         if (!right_vsl || ptr_list_size((struct ptr_list *)right
2100                                 goto free;

2102                         right_vs = first_ptr_list((struct ptr_list *)right_vsl);
2103                         if (strcmp(right_vs->var, right_name) != 0)
2104                                 goto free;
2105                         right_name = get_printed_param_name(expr, right_vs->var,
2106                         if (!right_name)
```

```
2107                                goto free;
2108                        snprintf(info_buf, sizeof(info_buf), "%s %s", show_speci
2109                        sql_insert_caller_info(expr, PARAM_COMPARE, i, "$", info

2111 free:
2112                        free_string(arg_name);
2113                } END_FOR_EACH_PTR(link);
2114        } END_FOR_EACH_PTR(arg);
2115 }

2117 static void struct_member_callback(struct expression *call, int param, char *pri
2118 {
2119         struct sm_state *compare_sm;
2120         struct string_list *links;
2121         char *link;
2122         struct compare_data *data;
2123         struct var_sym *left, *right;
2124         static char info_buf[256];
2125         const char *right_name;

2127         if (strstr(printed_name, " orig"))
2128                 return;

2130         links = link_sm->state->data;
2131         FOR_EACH_PTR(links, link) {
2132                 compare_sm = get_sm_state(compare_id, link, NULL);
2133                 if (!compare_sm)
2134                         continue;
2135                 data = compare_sm->state->data;
2136                 if (!data || !data->comparison)
2137                         continue;

2139                 if (ptr_list_size((struct ptr_list *)data->left_vsl) != 1 ||
2140                     ptr_list_size((struct ptr_list *)data->right_vsl) != 1)
2141                         continue;
2142                 left = first_ptr_list((struct ptr_list *)data->left_vsl);
2143                 right = first_ptr_list((struct ptr_list *)data->right_vsl);
2144                 if (left->sym == right->sym &&
2145                     strcmp(left->var, right->var) == 0)
2146                         continue;
2147                 /*
2148                  * Both parameters link to this comparison so only
2149                  * record the first one.
2150                  */
2151                 if (left->sym != link_sm->sym ||
2152                     strcmp(left->var, link_sm->name) != 0)
2153                         continue;

2155                 right_name = get_printed_param_name(call, right->var, right->sym
2156                 if (!right_name)
2157                         continue;
2158                 snprintf(info_buf, sizeof(info_buf), "%s %s", show_special(data-
2159                 sql_insert_caller_info(call, PARAM_COMPARE, param, printed_name,
2160         } END_FOR_EACH_PTR(link);
2161 }

2163 static void print_return_value_comparison(int return_id, char *return_ranges, st
2164 {
2165         char *name;
2166         const char *tmp_name;
2167         struct symbol *sym;
2168         int param;
2169         char info_buf[256];

2171         /*
2172          * TODO: This only prints == comparisons. That's probably the most
```



```
2173          * useful comparison because == max has lots of implications.  But it
2174          * would be good to capture the rest as well.
2175          *
2176          * This information is already in the DB but it's in the parameter math
2177          * bits and it's awkward to use it.  This is is the simpler, possibly
2178          * cleaner way, but not necessarily the best, I don't know.
2179          */

2181         if (!expr)
2182                 return;
2183         name = expr_to_var_sym(expr, &sym);
2184         if (!name || !sym)
2185                 goto free;

2187         param = get_param_num_from_sym(sym);
2188         if (param < 0)
2189                 goto free;
2190         if (param_was_set_var_sym(name, sym))
2191                 goto free;

2193         tmp_name = get_param_name_var_sym(name, sym);
2194         if (!tmp_name)
2195                 goto free;

2197         snprintf(info_buf, sizeof(info_buf), "== $%d%s", param, tmp_name + 1);
2198         sql_insert_return_states(return_id, return_ranges,
2199                                  PARAM_COMPARE, -1, "$", info_buf);
2200 free:
2201         free_string(name);
2202 }

2204 static void print_return_comparison(int return_id, char *return_ranges, struct e
2205 {
2206         struct sm_state *tmp;
2207         struct string_list *links;
2208         char *link;
2209         struct sm_state *sm;
2210         struct compare_data *data;
2211         struct var_sym *left, *right;
2212         int left_param, right_param;
2213         char left_buf[256];
2214         char right_buf[256];
2215         char info_buf[258];
2216         const char *tmp_name;

2218         print_return_value_comparison(return_id, return_ranges, expr);

2220         FOR_EACH_MY_SM(link_id, __get_cur_stree(), tmp) {
2221                 if (get_param_num_from_sym(tmp->sym) < 0)
2222                         continue;
2223                 links = tmp->state->data;
2224                 FOR_EACH_PTR(links, link) {
2225                         sm = get_sm_state(compare_id, link, NULL);
2226                         if (!sm)
2227                                 continue;
2228                         data = sm->state->data;
2229                         if (!data || !data->comparison)
2230                                 continue;
2231                         if (ptr_list_size((struct ptr_list *)data->left_vsl) !=
2232                             ptr_list_size((struct ptr_list *)data->right_vsl) !=
2233                                 continue;
2234                         left = first_ptr_list((struct ptr_list *)data->left_vsl)
2235                         right = first_ptr_list((struct ptr_list *)data->right_vs
2236                         if (left->sym == right->sym &&
2237                             strcmp(left->var, right->var) == 0)
2238                                 continue;
```

```
2239                        /*
2240                         * Both parameters link to this comparison so only
2241                         * record the first one.
2242                         */
2243                        if (left->sym != tmp->sym ||
2244                            strcmp(left->var, tmp->name) != 0)
2245                                continue;

2247                        if (strstr(right->var, " orig"))
2248                                continue;

2250                        left_param = get_param_num_from_sym(left->sym);
2251                        right_param = get_param_num_from_sym(right->sym);
2252                        if (left_param < 0 || right_param < 0)
2253                                continue;

2255                        tmp_name = get_param_name_var_sym(left->var, left->sym);
2256                        if (!tmp_name)
2257                                continue;
2258                        snprintf(left_buf, sizeof(left_buf), "%s", tmp_name);

2260                        tmp_name = get_param_name_var_sym(right->var, right->sym
2261                        if (!tmp_name || tmp_name[0] != '$')
2262                                continue;
2263                        snprintf(right_buf, sizeof(right_buf), "$%d%s", right_pa

2265                        /*
2266                         * FIXME: this should reject $ type variables (as
2267                         * opposed to $->foo type).  Those should come from
2268                         * smatch_param_compare_limit.c.
2269                         */

2271                        snprintf(info_buf, sizeof(info_buf), "%s %s", show_speci
2272                        sql_insert_return_states(return_id, return_ranges,
2273                                                 PARAM_COMPARE, left_param, left_buf, inf
2274                } END_FOR_EACH_PTR(link);

2276        } END_FOR_EACH_SM(tmp);
2277 }

2279 static int parse_comparison(char **value, int *op)
2280 {

2282         *op = **value;

2284         switch (*op) {
2285         case '<':
2286                 (*value)++;
2287                 if (**value == '=') {
2288                         (*value)++;
2289                         *op = SPECIAL_LTE;
2290                 }
2291                 break;
2292         case '=':
2293                 (*value)++;
2294                 (*value)++;
2295                 *op = SPECIAL_EQUAL;
2296                 break;
2297         case '!':
2298                 (*value)++;
2299                 (*value)++;
2300                 *op = SPECIAL_NOTEQUAL;
2301                 break;
2302         case '>':
2303                 (*value)++;
2304                 if (**value == '=') {
```



```
2305                         (*value)++;
2306                         *op = SPECIAL_GTE;
2307                 }
2308                 break;
2309         default:
2310                 return 0;
2311         }

2313         if (**value != ' ') {
2314                 sm_perror("parsing comparison.  %s", *value);
2315                 return 0;
2316         }

2318         (*value)++;
2319         return 1;
2320 }

2322 static int split_op_param_key(char *value, int *op, int *param, char **key)
2323 {
2324         static char buf[256];
2325         char *p;

2327         if (!parse_comparison(&value, op))
2328                 return 0;

2330         snprintf(buf, sizeof(buf), value);

2332         p = buf;
2333         if (*p++ != '$')
2334                 return 0;

2336         *param = atoi(p);
2337         if (*param < 0 || *param > 99)
2338                 return 0;
2339         p++;
2340         if (*param > 9)
2341                 p++;
2342         p--;
2343         *p = '$';
2344         *key = p;

2346         return 1;
2347 }

2349 static void db_return_comparison(struct expression *expr, int left_param, char *
2350 {
2351         struct expression *left_arg, *right_arg;
2352         char *left_name = NULL;
2353         struct symbol *left_sym;
2354         char *right_name = NULL;
2355         struct symbol *right_sym;
2356         int op;
2357         int right_param;
2358         char *right_key;
2359         struct var_sym_list *left_vsl = NULL, *right_vsl = NULL;

2361         if (left_param == -1) {
2362                 if (expr->type != EXPR_ASSIGNMENT)
2363                         return;
2364                 left_arg = strip_expr(expr->left);

2366                 while (expr->type == EXPR_ASSIGNMENT)
2367                         expr = strip_expr(expr->right);
2368                 if (expr->type != EXPR_CALL)
2369                         return;
2370         } else {
```

```
2371                while (expr->type == EXPR_ASSIGNMENT)
2372                        expr = strip_expr(expr->right);
2373                if (expr->type != EXPR_CALL)
2374                        return;
2376                left_arg = get_argument_from_call_expr(expr->args, left_param);
2377                if (!left_arg)
2378                        return;
2379        }
2381        if (!split_op_param_key(value, &op, &right_param, &right_key))
2382                return;
2384        right_arg = get_argument_from_call_expr(expr->args, right_param);
2385        if (!right_arg)
2386                return;
2388        left_name = get_variable_from_key(left_arg, key, &left_sym);
2389        if (!left_name || !left_sym)
2390                goto free;
2392        right_name = get_variable_from_key(right_arg, right_key, &right_sym);
2393        if (!right_name || !right_sym)
2394                goto free;
2396        add_var_sym(&left_vsl, left_name, left_sym);
2397        add_var_sym(&right_vsl, right_name, right_sym);
2399        add_comparison_var_sym(NULL, left_name, left_vsl, op, NULL, right_name,
2401 free:
2402        free_string(left_name);
2403        free_string(right_name);
2404 }
2406 int param_compare_limit_is_impossible(struct expression *expr, int left_param, c
2407 {
2408        struct smatch_state *state;
2409        char *left_name = NULL;
2410        char *right_name = NULL;
2411        struct symbol *left_sym, *right_sym;
2412        struct expression *left_arg, *right_arg;
2413        int op, state_op;
2414        int right_param;
2415        char *right_key;
2416        int ret = 0;
2417        char buf[256];
2419        while (expr->type == EXPR_ASSIGNMENT)
2420                expr = strip_expr(expr->right);
2421        if (expr->type != EXPR_CALL)
2422                return 0;
2424        if (!split_op_param_key(value, &op, &right_param, &right_key))
2425                return 0;
2427        left_arg = get_argument_from_call_expr(expr->args, left_param);
2428        if (!left_arg)
2429                return 0;
2431        right_arg = get_argument_from_call_expr(expr->args, right_param);
2432        if (!right_arg)
2433                return 0;
2435        left_name = get_variable_from_key(left_arg, left_key, &left_sym);
2436        right_name = get_variable_from_key(right_arg, right_key, &right_sym);
```



```
2437        if (!left_name || !right_name)
2438                goto free;
2440        snprintf(buf, sizeof(buf), "%s vs %s", left_name, right_name);
2441        state = get_state(compare_id, buf, NULL);
2442        if (!state)
2443                goto free;
2444        state_op = state_to_comparison(state);
2445        if (!state_op)
2446                goto free;
2448        if (!filter_comparison(remove_unsigned_from_comparison(state_op), op))
2449                ret = 1;
2450 free:
2451        free_string(left_name);
2452        free_string(right_name);
2453        return ret;
2454 }
2456 int impossibly_high_comparison(struct expression *expr)
2457 {
2458        struct smatch_state *link_state;
2459        struct sm_state *sm;
2460        struct string_list *links;
2461        char *link;
2462        struct compare_data *data;
2464        link_state = get_state_expr(link_id, expr);
2465        if (!link_state) {
2466                if (expr->type == EXPR_BINOP &&
2467                    (impossibly_high_comparison(expr->left) ||
2468                     impossibly_high_comparison(expr->right)))
2469                        return 1;
2470                return 0;
2471        }
2473        links = link_state->data;
2474        FOR_EACH_PTR(links, link) {
2475                sm = get_sm_state(compare_id, link, NULL);
2476                if (!sm)
2477                        continue;
2478                data = sm->state->data;
2479                if (!data)
2480                        continue;
2481                if (!possibly_true(data->left, data->comparison, data->right))
2482                        return 1;
2483        } END_FOR_EACH_PTR(link);
2485        return 0;
2486 }
2488 static void free_data(struct symbol *sym)
2489 {
2490        if (__inline_fn)
2491                return;
2492        clear_compare_data_alloc();
2493 }
2495 void register_comparison(int id)
2496 {
2497        compare_id = id;
2498        add_hook(&save_start_states, AFTER_DEF_HOOK);
2499        add_unmatched_state_hook(compare_id, unmatched_comparison);
2500        add_pre_merge_hook(compare_id, &pre_merge_hook);
2501        add_merge_hook(compare_id, &merge_compare_states);
2502        add_hook(&free_data, AFTER_FUNC_HOOK);
```

```
2503         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
2504         add_split_return_callback(&print_return_comparison);

2506         select_return_states_hook(PARAM_COMPARE, &db_return_comparison);
2507         add_hook(&match_preop, OP_HOOK);
2508 }

2510 void register_comparison_late(int id)
2511 {
2512         add_hook(&match_assign, ASSIGNMENT_HOOK);
2513 }

2515 void register_comparison_links(int id)
2516 {
2517         link_id = id;
2518         add_merge_hook(link_id, &merge_links);
2519         add_modification_hook(link_id, &match_modify);
2520         add_modification_hook_late(link_id, match_inc_dec);

2522         add_member_info_callback(link_id, struct_member_callback);
2523 }

2525 void register_comparison_inc_dec(int id)
2526 {
2527         inc_dec_id = id;
2528         add_modification_hook_late(inc_dec_id, &iter_modify);
2529 }

2531 void register_comparison_inc_dec_links(int id)
2532 {
2533         inc_dec_link_id = id;
2534         set_up_link_functions(inc_dec_id, inc_dec_link_id);
2535 }

2537 static void filter_by_sm(struct sm_state *sm, int op,
2538                          struct state_list **true_stack,
2539                          struct state_list **false_stack)
2540 {
2541         struct compare_data *data;
2542         int istrue = 0;
2543         int isfalse = 0;

2545         if (!sm)
2546                 return;
2547         data = sm->state->data;
2548         if (!data) {
2549                 if (sm->merged) {
2550                         filter_by_sm(sm->left, op, true_stack, false_stack);
2551                         filter_by_sm(sm->right, op, true_stack, false_stack);
2552                 }
2553                 return;
2554         }

2556         if (data->comparison &&
2557             data->comparison == filter_comparison(data->comparison, op))
2558                 istrue = 1;

2560         if (data->comparison &&
2561             data->comparison == filter_comparison(data->comparison, negate_compa
2562                 isfalse = 1;

2564         if (istrue)
2565                 add_ptr_list(true_stack, sm);
2566         if (isfalse)
2567                 add_ptr_list(false_stack, sm);
```



```
2569         if (sm->merged) {
2570                 filter_by_sm(sm->left, op, true_stack, false_stack);
2571                 filter_by_sm(sm->right, op, true_stack, false_stack);
2572         }
2573 }

2575 struct sm_state *comparison_implication_hook(struct expression *expr,
2576                                              struct state_list **true_stack,
2577                                              struct state_list **false_stack)
2578 {
2579         struct sm_state *sm;
2580         char *left, *right;
2581         int op;
2582         static char buf[256];

2584         if (expr->type != EXPR_COMPARE)
2585                 return NULL;

2587         op = expr->op;

2589         left = expr_to_var(expr->left);
2590         right = expr_to_var(expr->right);
2591         if (!left || !right) {
2592                 free_string(left);
2593                 free_string(right);
2594                 return NULL;
2595         }

2597         if (strcmp(left, right) > 0) {
2598                 char *tmp = left;

2600                 left = right;
2601                 right = tmp;
2602                 op = flip_comparison(op);
2603         }

2605         snprintf(buf, sizeof(buf), "%s vs %s", left, right);
2606         sm = get_sm_state(compare_id, buf, NULL);
2607         if (!sm)
2608                 return NULL;
2609         if (!sm->merged)
2610                 return NULL;

2612         filter_by_sm(sm, op, true_stack, false_stack);
2613         if (!*true_stack && !*false_stack)
2614                 return NULL;

2616         if (option_debug)
2617                 sm_msg("implications from comparison: (%s)", show_sm(sm));

2619         return sm;
2620 }
```

```
**********************************************************
   19097 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_conditions.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2006,2008 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The simplest type of condition is
 * if (a) { ...
 *
 * The next simplest kind of conditions is
 * if (a && b) { c;
 * In that case 'a' is true when we get to 'b' and both are true
 * when we get to c.
 *
 * Or's are a little more complicated.
 * if (a || b) { c;
 * We know 'a' is not true when we get to 'b' but it may be true
 * when we get to c.
 *
 * If we mix and's and or's that's even more complicated.
 * if (a && b && c || a && d) { d ;
 * 'a' is true when we evaluate 'b', and 'd'.
 * 'b' is true when we evaluate 'c' but otherwise we don't.
 *
 * The other thing that complicates matters is if we negate
 * some if conditions.
 * if (!a) { ...
 * Smatch has passes the un-negated version to the client and flip
 * the true and false values internally.  This makes it easier
 * to write checks.
 *
 * And negations can be part of a compound.
 * if (a && !(b || c)) { d;
 * In that situation we multiply the negative through to simplify
 * stuff so that we can remove the parens like this:
 * if (a && !b && !c) { d;
 *
 * One other thing is that:
 * if ((a) != 0){ ...
 * that's basically the same as testing for just 'a' and we simplify
 * comparisons with zero before passing it to the script.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"
#include "smatch_expression_stacks.h"
```



```
extern int __expr_stmt_count;

struct expression_list *big_condition_stack;

static void split_conditions(struct expression *expr);

static int is_logical_and(struct expression *expr)
{
        if (expr->op == SPECIAL_LOGICAL_AND)
                return 1;
        return 0;
}

static int handle_zero_comparisons(struct expression *expr)
{
        struct expression *tmp = NULL;
        struct expression *zero;

        // if left is zero or right is zero
        if (is_zero(expr->left)) {
                zero = strip_expr(expr->left);
                if (zero->type != EXPR_VALUE)
                        __split_expr(expr->left);
                tmp = expr->right;
        } else if (is_zero(expr->right)) {
                zero = strip_expr(expr->left);
                if (zero->type != EXPR_VALUE)
                        __split_expr(expr->right);
                tmp = expr->left;
        } else {
                return 0;
        }

        // "if (foo != 0)" is the same as "if (foo)"
        if (expr->op == SPECIAL_NOTEQUAL) {
                split_conditions(tmp);
                return 1;
        }

        // "if (foo == 0)" is the same as "if (!foo)"
        if (expr->op == SPECIAL_EQUAL) {
                split_conditions(tmp);
                __negate_cond_stacks();
                return 1;
        }

        return 0;
}

/*
 * This function is for handling calls to likely/unlikely
 */

static int ignore_builtin_expect(struct expression *expr)
{
        if (sym_name_is("__builtin_expect", expr->fn)) {
                split_conditions(first_ptr_list((struct ptr_list *) expr->args));
                return 1;
        }
        return 0;
}

/*
 * handle_compound_stmt() is for: foo = ({blah; blah; blah; 1})
 */
```

```
128 static void handle_compound_stmt(struct statement *stmt)
129 {
130         struct expression *expr = NULL;
131         struct statement *last;
132         struct statement *s;
133
134         last = last_ptr_list((struct ptr_list *)stmt->stmts);
135         if (last->type == STMT_LABEL) {
136                 if (last->label_statement &&
137                     last->label_statement->type == STMT_EXPRESSION)
138                         expr = last->label_statement->expression;
139                 else
140                         last = NULL;
141         } else if (last->type != STMT_EXPRESSION) {
142                 last = NULL;
143         } else {
144                 expr = last->expression;
145         }
146
147         FOR_EACH_PTR(stmt->stmts, s) {
148                 if (s != last)
149                         __split_stmt(s);
150         } END_FOR_EACH_PTR(s);
151         if (last && last->type == STMT_LABEL)
152                 __split_label_stmt(last);
153         split_conditions(expr);
154 }
155
156 static int handle_preop(struct expression *expr)
157 {
158         struct statement *stmt;
159
160         if (expr->op == '!') {
161                 split_conditions(expr->unop);
162                 __negate_cond_stacks();
163                 return 1;
164         }
165         stmt = get_expression_statement(expr);
166         if (stmt) {
167                 handle_compound_stmt(stmt);
168                 return 1;
169         }
170         return 0;
171 }
172
173 static void handle_logical(struct expression *expr)
174 {
175         /*
176          * If we come to an "and" expr then:
177          * We split the left side.
178          * We keep all the current states.
179          * We split the right side.
180          * We keep all the states from both true sides.
181          *
182          * If it's an "or" expr then:
183          * We save the current slist.
184          * We split the left side.
185          * We use the false states for the right side.
186          * We split the right side.
187          * We save all the states that are the same on both sides.
188          */
189
190         split_conditions(expr->left);
191
192         if (is_logical_and(expr))
```



```
193                 __use_cond_true_states();
194         else
195                 __use_cond_false_states();
196
197         __push_cond_stacks();
198
199         __save_pre_cond_states();
200         split_conditions(expr->right);
201         __discard_pre_cond_states();
202
203         if (is_logical_and(expr))
204                 __and_cond_states();
205         else
206                 __or_cond_states();
207
208         __use_cond_true_states();
209 }
210
211 static struct stree *combine_strees(struct stree *orig, struct stree *fake, stru
212 {
213         struct stree *ret = NULL;
214
215         overwrite_stree(orig, &ret);
216         overwrite_stree(fake, &ret);
217         overwrite_stree(new, &ret);
218         free_stree(&new);
219
220         return ret;
221 }
222
223 /*
224  * handle_select()
225  * if ((aaa()?bbb():ccc())) { ...
226  *
227  * This is almost the same as:
228  * if ((aaa() && bbb()) || (!aaa() && ccc())) { ...
229  *
230  * It's a bit complicated because we shouldn't pass aaa()
231  * to the clients more than once.
232  */
233
234 static void handle_select(struct expression *expr)
235 {
236         struct stree *a_T = NULL;
237         struct stree *a_F = NULL;
238         struct stree *a_T_b_T = NULL;
239         struct stree *a_T_b_F = NULL;
240         struct stree *a_T_b_fake = NULL;
241         struct stree *a_F_c_T = NULL;
242         struct stree *a_F_c_F = NULL;
243         struct stree *a_F_c_fake = NULL;
244         struct stree *tmp;
245         struct sm_state *sm;
246
247         /*
248          * Imagine we have this:  if (a ? b : c) { ...
249          *
250          * The condition is true if "a" is true and "b" is true or
251          * "a" is false and "c" is true.  It's false if "a" is true
252          * and "b" is false or "a" is false and "c" is false.
253          *
254          * The variable name "a_T_b_T" stands for "a true b true" etc.
255          *
256          * But if we know "b" is true then we can simpilify things.
257          * The condition is true if "a" is true or if "a" is false and
258          * "c" is true.  The only way the condition can be false is if
```

```
259          * "a" is false and "c" is false.
260          *
261          * The remaining thing is the "a_T_b_fake".  When we simplify
262          * the equations we have to take into consideration that other
263          * states may have changed that don't play into the true false
264          * equation.  Take the following example:
265          * if ({
266          *         (flags) = __raw_local_irq_save();
267          *         _spin_trylock(lock) ? 1 :
268          *                 ({ raw_local_irq_restore(flags);  0; });
269          *    })
270          * Smatch has to record that the irq flags were restored on the
271          * false path.
272          *
273          */

275         __save_pre_cond_states();

277         split_conditions(expr->conditional);

279         a_T = __copy_cond_true_states();
280         a_F = __copy_cond_false_states();

282         __use_cond_true_states();

284         __push_cond_stacks();
285         __push_fake_cur_stree();
286         split_conditions(expr->cond_true);
287         __process_post_op_stack();
288         a_T_b_fake = __pop_fake_cur_stree();
289         a_T_b_T = combine_strees(a_T, a_T_b_fake, __pop_cond_true_stack());
290         a_T_b_F = combine_strees(a_T, a_T_b_fake, __pop_cond_false_stack());

292         __use_cond_false_states();

294         __push_cond_stacks();
295         __push_fake_cur_stree();
296         split_conditions(expr->cond_false);
297         a_F_c_fake = __pop_fake_cur_stree();
298         a_F_c_T = combine_strees(a_F, a_F_c_fake, __pop_cond_true_stack());
299         a_F_c_F = combine_strees(a_F, a_F_c_fake, __pop_cond_false_stack());

301         /* We have to restore the pre condition states so that
302            implied_condition_true() will use the right cur_stree */
303         __use_pre_cond_states();

305         if (implied_condition_true(expr->cond_true)) {
306                 free_stree(&a_T_b_T);
307                 free_stree(&a_T_b_F);
308                 a_T_b_T = clone_stree(a_T);
309                 overwrite_stree(a_T_b_fake, &a_T_b_T);
310         }
311         if (implied_condition_false(expr->cond_true)) {
312                 free_stree(&a_T_b_T);
313                 free_stree(&a_T_b_F);
314                 a_T_b_F = clone_stree(a_T);
315                 overwrite_stree(a_T_b_fake, &a_T_b_F);
316         }
317         if (implied_condition_true(expr->cond_false)) {
318                 free_stree(&a_F_c_T);
319                 free_stree(&a_F_c_F);
320                 a_F_c_T = clone_stree(a_F);
321                 overwrite_stree(a_F_c_fake, &a_F_c_T);
322         }
323         if (implied_condition_false(expr->cond_false)) {
324                 free_stree(&a_F_c_T);
```



```
325                 free_stree(&a_F_c_F);
326                 a_F_c_F = clone_stree(a_F);
327                 overwrite_stree(a_F_c_fake, &a_F_c_F);
328         }

330         merge_stree(&a_T_b_T, a_F_c_T);
331         merge_stree(&a_T_b_F, a_F_c_F);

333         tmp = __pop_cond_true_stack();
334         free_stree(&tmp);
335         tmp = __pop_cond_false_stack();
336         free_stree(&tmp);

338         __push_cond_stacks();
339         FOR_EACH_SM(a_T_b_T, sm) {
340                 __set_true_false_sm(sm, NULL);
341         } END_FOR_EACH_SM(sm);
342         FOR_EACH_SM(a_T_b_F, sm) {
343                 __set_true_false_sm(NULL, sm);
344         } END_FOR_EACH_SM(sm);
345         __free_set_states();

347         free_stree(&a_T_b_fake);
348         free_stree(&a_F_c_fake);
349         free_stree(&a_F_c_T);
350         free_stree(&a_F_c_F);
351         free_stree(&a_T_b_T);
352         free_stree(&a_T_b_F);
353         free_stree(&a_T);
354         free_stree(&a_F);
355 }

357 static void handle_comma(struct expression *expr)
358 {
359         __split_expr(expr->left);
360         split_conditions(expr->right);
361 }

363 static int make_op_unsigned(int op)
364 {
365         switch (op) {
366         case '<':
367                 return SPECIAL_UNSIGNED_LT;
368         case SPECIAL_LTE:
369                 return SPECIAL_UNSIGNED_LTE;
370         case '>':
371                 return SPECIAL_UNSIGNED_GT;
372         case SPECIAL_GTE:
373                 return SPECIAL_UNSIGNED_GTE;
374         }
375         return op;
376 }

378 static void hackup_unsigned_compares(struct expression *expr)
379 {
380         if (expr->type != EXPR_COMPARE)
381                 return;

383         if (type_unsigned(get_type(expr)))
384                 expr->op = make_op_unsigned(expr->op);
385 }

387 static void do_condition(struct expression *expr)
388 {
389         __fold_in_set_states();
390         __push_fake_cur_stree();
```

```
391         __pass_to_client(expr, CONDITION_HOOK);
392         __fold_in_set_states();
393 }

395 static void split_conditions(struct expression *expr)
396 {
397         if (option_debug) {
398                 char *cond = expr_to_str(expr);

400                 sm_msg("%d in split_conditions (%s)", get_lineno(), cond);
401                 free_string(cond);
402         }

404         expr = strip_expr_set_parent(expr);
405         if (!expr) {
406                 __fold_in_set_states();
407                 return;
408         }

410         /*
411          * On fast paths (and also I guess some people think it's cool) people
412          * sometimes use | instead of ||.  It works the same basically except
413          * that || implies a memory barrier between conditions.  The easiest way
414          * to handle it is by pretending that | also has a barrier and re-using
415          * all the normal condition code.  This potentially hides some bugs, but
416          * people who write code like this should just be careful or they
417          * deserve bugs.
418          *
419          * We could potentially treat boolean bitwise & this way but that seems
420          * too complicated to deal with.
421          */
422         if (expr->type == EXPR_BINOP && expr->op == '|') {
423                 handle_logical(expr);
424                 return;
425         }

427         switch (expr->type) {
428         case EXPR_LOGICAL:
429                 expr_set_parent_expr(expr->left, expr);
430                 expr_set_parent_expr(expr->right, expr);
431                 __pass_to_client(expr, LOGIC_HOOK);
432                 handle_logical(expr);
433                 return;
434         case EXPR_COMPARE:
435                 expr_set_parent_expr(expr->left, expr);
436                 expr_set_parent_expr(expr->right, expr);
437                 hackup_unsigned_compares(expr);
438                 if (handle_zero_comparisons(expr))
439                         return;
440                 break;
441         case EXPR_CALL:
442                 if (ignore_builtin_expect(expr))
443                         return;
444                 break;
445         case EXPR_PREOP:
446                 expr_set_parent_expr(expr->unop, expr);
447                 if (handle_preop(expr))
448                         return;
449                 break;
450         case EXPR_CONDITIONAL:
451         case EXPR_SELECT:
452                 expr_set_parent_expr(expr->conditional, expr);
453                 expr_set_parent_expr(expr->cond_true, expr);
454                 expr_set_parent_expr(expr->cond_false, expr);
455                 handle_select(expr);
456                 return;
```



```
457         case EXPR_COMMA:
458                 expr_set_parent_expr(expr->left, expr);
459                 expr_set_parent_expr(expr->right, expr);
460                 handle_comma(expr);
461                 return;
462         }

464         /* fixme: this should be in smatch_flow.c
465            but because of the funny stuff we do with conditions
466            it's awkward to put it there.  We would need to
467            call CONDITION_HOOK in smatch_flow as well.
468         */
469         push_expression(&big_expression_stack, expr);
470         push_expression(&big_condition_stack, expr);

472         if (expr->type == EXPR_COMPARE) {
473                 if (expr->left->type != EXPR_POSTOP)
474                         __split_expr(expr->left);
475                 if (expr->right->type != EXPR_POSTOP)
476                         __split_expr(expr->right);
477         } else if (expr->type != EXPR_POSTOP) {
478                 __split_expr(expr);
479         }
480         do_condition(expr);
481         if (expr->type == EXPR_COMPARE) {
482                 if (expr->left->type == EXPR_POSTOP)
483                         __split_expr(expr->left);
484                 if (expr->right->type == EXPR_POSTOP)
485                         __split_expr(expr->right);
486         } else if (expr->type == EXPR_POSTOP) {
487                 __split_expr(expr);
488         }
489         __push_fake_cur_stree();
490         __process_post_op_stack();
491         __fold_in_set_states();
492         pop_expression(&big_condition_stack);
493         pop_expression(&big_expression_stack);
494 }

496 static int inside_condition;
497 void __split_whole_condition(struct expression *expr)
498 {
499         sm_debug("%d in __split_whole_condition\n", get_lineno());
500         inside_condition++;
501         __save_pre_cond_states();
502         __push_cond_stacks();
503         /* it's a hack, but it's sometimes handy to have this stuff
504            on the big_expression_stack.  */
505         push_expression(&big_expression_stack, expr);
506         split_conditions(expr);
507         __use_cond_states();
508         __pass_to_client(expr, WHOLE_CONDITION_HOOK);
509         pop_expression(&big_expression_stack);
510         inside_condition--;
511         sm_debug("%d done __split_whole_condition\n", get_lineno());
512 }

514 void __handle_logic(struct expression *expr)
515 {
516         sm_debug("%d in __handle_logic\n", get_lineno());
517         inside_condition++;
518         __save_pre_cond_states();
519         __push_cond_stacks();
520         /* it's a hack, but it's sometimes handy to have this stuff
521            on the big_expression_stack.  */
522         push_expression(&big_expression_stack, expr);
```

```
523         if (expr)
524                 split_conditions(expr);
525         __use_cond_states();
526         __pass_to_client(expr, WHOLE_CONDITION_HOOK);
527         pop_expression(&big_expression_stack);
528         __merge_false_states();
529         inside_condition--;
530         sm_debug("%d done __handle_logic\n", get_lineno());
531 }

533 int is_condition(struct expression *expr)
534 {

536         expr = strip_expr(expr);
537         if (!expr)
538                 return 0;

540         switch (expr->type) {
541         case EXPR_LOGICAL:
542         case EXPR_COMPARE:
543                 return 1;
544         case EXPR_PREOP:
545                 if (expr->op == '!')
546                         return 1;
547         }
548         return 0;
549 }

551 int __handle_condition_assigns(struct expression *expr)
552 {
553         struct expression *right;
554         struct stree *true_stree, *false_stree, *fake_stree;
555         struct sm_state *sm;

557         if (expr->op != '=')
558                 return 0;
559         right = strip_expr(expr->right);
560         if (!is_condition(expr->right))
561                 return 0;

563         sm_debug("%d in __handle_condition_assigns\n", get_lineno());
564         inside_condition++;
565         __save_pre_cond_states();
566         __push_cond_stacks();
567         /* it's a hack, but it's sometimes handy to have this stuff
568            on the big_expression_stack.  */
569         push_expression(&big_expression_stack, right);
570         split_conditions(right);
571         true_stree = __get_true_states();
572         false_stree = __get_false_states();
573         __use_cond_states();
574         __push_fake_cur_stree();
575         set_extra_expr_mod(expr->left, alloc_estate_sval(sval_type_val(get_type(
576         __pass_to_client(right, WHOLE_CONDITION_HOOK);

578         fake_stree = __pop_fake_cur_stree();
579         FOR_EACH_SM(fake_stree, sm) {
580                 overwrite_sm_state_stree(&true_stree, sm);
581         } END_FOR_EACH_SM(sm);
582         free_stree(&fake_stree);

584         pop_expression(&big_expression_stack);
585         inside_condition--;

587         __push_true_states();
```



```
589         __use_false_states();
590         __push_fake_cur_stree();
591         set_extra_expr_mod(expr->left, alloc_estate_sval(sval_type_val(get_type(

593         fake_stree = __pop_fake_cur_stree();
594         FOR_EACH_SM(fake_stree, sm) {
595                 overwrite_sm_state_stree(&false_stree, sm);
596         } END_FOR_EACH_SM(sm);
597         free_stree(&fake_stree);

599         __merge_true_states();
600         merge_fake_stree(&true_stree, false_stree);
601         free_stree(&false_stree);
602         FOR_EACH_SM(true_stree, sm) {
603                 __set_sm(sm);
604         } END_FOR_EACH_SM(sm);

606         __pass_to_client(expr, ASSIGNMENT_HOOK);
607         sm_debug("%d done __handle_condition_assigns\n", get_lineno());
608         return 1;
609 }

611 static int is_select_assign(struct expression *expr)
612 {
613         struct expression *right;

615         if (expr->op != '=')
616                 return 0;
617         right = strip_expr(expr->right);
618         if (right->type == EXPR_CONDITIONAL)
619                 return 1;
620         if (right->type == EXPR_SELECT)
621                 return 1;
622         return 0;
623 }

625 int __handle_select_assigns(struct expression *expr)
626 {
627         struct expression *right;
628         struct stree *final_states = NULL;
629         struct sm_state *sm;
630         int is_true;
631         int is_false;

633         if (!is_select_assign(expr))
634                 return 0;
635         sm_debug("%d in __handle_ternary_assigns\n", get_lineno());
636         right = strip_expr(expr->right);
637         __pass_to_client(right, SELECT_HOOK);

639         is_true = implied_condition_true(right->conditional);
640         is_false = implied_condition_false(right->conditional);

642         /* hah hah.  the ultra fake out */
643         __save_pre_cond_states();
644         __split_whole_condition(right->conditional);

646         if (!is_false) {
647                 struct expression *fake_expr;

649                 if (right->cond_true)
650                         fake_expr = assign_expression(expr->left, expr->op, righ
651                 else
652                         fake_expr = assign_expression(expr->left, expr->op, righ
653                 __split_expr(fake_expr);
654                 final_states = clone_stree(__get_cur_stree());
```

```
655         }

657         __use_false_states();
658         if (!is_true) {
659                 struct expression *fake_expr;

661                 fake_expr = assign_expression(expr->left, expr->op, right->cond_
662                 __split_expr(fake_expr);
663                 merge_stree(&final_states, __get_cur_stree());
664         }

666         __use_pre_cond_states();

668         FOR_EACH_SM(final_states, sm) {
669                 __set_sm(sm);
670         } END_FOR_EACH_SM(sm);

672         free_stree(&final_states);

674         sm_debug("%d done __handle_ternary_assigns\n", get_lineno());

676         return 1;
677 }

679 static struct statement *split_then_return_last(struct statement *stmt)
680 {
681         struct statement *tmp;
682         struct statement *last_stmt;

684         last_stmt = last_ptr_list((struct ptr_list *)stmt->stmts);
685         if (!last_stmt)
686                 return NULL;

688         __push_scope_hooks();
689         FOR_EACH_PTR(stmt->stmts, tmp) {
690                 if (tmp == last_stmt) {
691                         if (tmp->type == STMT_LABEL) {
692                                 __split_label_stmt(tmp);
693                                 return tmp->label_statement;
694                         }
695                         return last_stmt;
696                 }
697                 __split_stmt(tmp);
698         } END_FOR_EACH_PTR(tmp);
699         return NULL;
700 }

702 int __handle_expr_statement_assigns(struct expression *expr)
703 {
704         struct expression *right;
705         struct statement *stmt;

707         right = expr->right;
708         if (right->type == EXPR_PREOP && right->op == '(')
709                 right = right->unop;
710         if (right->type != EXPR_STATEMENT)
711                 return 0;

713         __expr_stmt_count++;
714         stmt = right->statement;
715         if (stmt->type == STMT_COMPOUND) {
716                 struct statement *last_stmt;
717                 struct expression *fake_assign;
718                 struct expression fake_expr_stmt = { .smatch_flags = Fake, };

720                 last_stmt = split_then_return_last(stmt);
```



```
721                 if (!last_stmt) {
722                         __expr_stmt_count--;
723                         return 0;
724                 }

726                 fake_expr_stmt.pos = last_stmt->pos;
727                 fake_expr_stmt.type = EXPR_STATEMENT;
728                 fake_expr_stmt.op = 0;
729                 fake_expr_stmt.statement = last_stmt;

731                 fake_assign = assign_expression(expr->left, expr->op, &fake_expr
732                 __split_expr(fake_assign);

734                 __pass_to_client(stmt, STMT_HOOK_AFTER);
735                 __call_scope_hooks();
736         } else if (stmt->type == STMT_EXPRESSION) {
737                 struct expression *fake_assign;

739                 fake_assign = assign_expression(expr->left, expr->op, stmt->expr
740                 __split_expr(fake_assign);

742         } else {
743                 __split_stmt(stmt);
744         }
745         __expr_stmt_count--;
746         return 1;
747 }

749 int in_condition(void)
750 {
751         return inside_condition;
752 }
```

```
**********************************************************
   13125 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_constraints.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Basically I see constraints as a way of saying "x <= some_limit".  The
 * problem is that smatch_capped is not granullar enough.
 *
 * This is mostly for finding out of bounds errors.  So there are different
 * types of constraints.  Quite often we have "foo->xxx[i] = 42;" and we want
 * to verify that "i" is less than foo->size.
 *
 * My idea was that we could automatically figure out these constraints.  And we
 * could load them in the DB so that they are the same every time.  As in a
 * constraint could be "< (struct whatever)->size" and give that in ID that
 * would be constant until you completely wiped the DB.  So when you do a normal
 * DB rebuild then the first thing it will do is preserve all the constraints.
 * I guess the reason to do it this way is to save space...  I sometimes suspect
 * that worrying about saving space is premature optimization.
 *
 * The other thing that I want to do a little bit different here is how I merge
 * constraints.  If a constraint is true on both sides, then that's normal.  If
 * we merge constraint 23 and 67 then we get constraint 23|67.  If we merge 23
 * with &undefined then we get &undefined.  We can also have two constraints
 * that are both true so we could have (45&23)|12 which means either both 45 and
 * 23 are true or 12 is true.
 *
 */


#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

ALLOCATOR(constraint, "constraints");

static void add_constraint(struct constraint_list **list, int op, int constraint
{
        struct constraint *tmp, *new;

        FOR_EACH_PTR(*list, tmp) {
                if (tmp->id < constraint)
                        continue;
                if (tmp->id == constraint) {
                        if (tmp->op == '<')
```



```
                                return;
                        if (op == SPECIAL_LTE)
                                return;

                        new = __alloc_constraint(0);
                        new->op = op;
                        new->id = constraint;
                        REPLACE_CURRENT_PTR(tmp, new);
                        return;
                }

                new = __alloc_constraint(0);
                new->op = op;
                new->id = constraint;
                INSERT_CURRENT(new, tmp);
                return;
        } END_FOR_EACH_PTR(tmp);

        new = __alloc_constraint(0);
        new->op = op;
        new->id = constraint;
        add_ptr_list(list, new);
}

static struct constraint_list *merge_constraint_lists(struct constraint_list *on
{
        struct constraint_list *ret = NULL;
        struct constraint *tmp;

        // FIXME: not || but &&
        FOR_EACH_PTR(one, tmp) {
                add_constraint(&ret, tmp->op, tmp->id);
        } END_FOR_EACH_PTR(tmp);

        FOR_EACH_PTR(two, tmp) {
                add_constraint(&ret, tmp->op, tmp->id);
        } END_FOR_EACH_PTR(tmp);

        return ret;
}

static struct constraint_list *clone_constraint_list(struct constraint_list *lis
{
        struct constraint_list *ret = NULL;
        struct constraint *tmp;

        FOR_EACH_PTR(list, tmp) {
                add_constraint(&ret, tmp->op, tmp->id);
        } END_FOR_EACH_PTR(tmp);

        return ret;
}

static struct smatch_state *alloc_constraint_state(struct constraint_list *list)
{
        struct smatch_state *state;
        struct constraint *con;
        static char buf[256];
        int cnt = 0;

        FOR_EACH_PTR(list, con) {
                if (cnt != 0)
                        cnt += snprintf(buf + cnt, sizeof(buf) - cnt, ", ");
                cnt += snprintf(buf + cnt, sizeof(buf) - cnt, "%s%d",
                                show_special(con->op), con->id);
        } END_FOR_EACH_PTR(con);
```

```
128         state = __alloc_smatch_state(0);
129         state->name = alloc_string(buf);
130         state->data = list;
131         return state;
132 }

134 static struct smatch_state *merge_func(struct smatch_state *s1, struct smatch_st
135 {
136         struct constraint_list *list;

138         // FIXME:  use the dead code below instead
139         if (strcmp(s1->name, s2->name) == 0)
140                 return s1;
141         return &merged;

143         list = merge_constraint_lists(s1->data, s2->data);
144         return alloc_constraint_state(list);
145 }

147 static int negate_gt(int op)
148 {
149         switch (op) {
150         case '>':
151         case SPECIAL_UNSIGNED_GT:
152         case SPECIAL_GTE:
153         case SPECIAL_UNSIGNED_GTE:
154                 return negate_comparison(op);
155         }
156         return op;
157 }

159 static char *get_func_constraint(struct expression *expr)
160 {
161         char buf[256];
162         char *name;

164         if (is_fake_call(expr))
165                 return NULL;
166         name = expr_to_str(expr->fn);
167         if (!name)
168                 return NULL;
169         snprintf(buf, sizeof(buf), "%s()", name);
170         free_string(name);
171         return alloc_string(buf);
172 }

174 static char *get_toplevel_name(struct expression *expr)
175 {
176         struct symbol *sym;
177         char buf[256];

179         expr = strip_expr(expr);
180         if (expr->type != EXPR_SYMBOL || !expr->symbol || !expr->symbol->ident)
181                 return NULL;

183         sym = expr->symbol;
184         if (!(sym->ctype.modifiers & MOD_TOPLEVEL))
185                 return NULL;

187         if (sym->ctype.modifiers & MOD_STATIC)
188                 snprintf(buf, sizeof(buf), "%s %s", get_base_file(), sym->ident-
189         else
190                 snprintf(buf, sizeof(buf), "extern %s", sym->ident->name);

192         return alloc_string(buf);
```



```
193 }

195 char *get_constraint_str(struct expression *expr)
196 {
197         char *name;

199         if (!expr)
200                 return NULL;
201         if (expr->type == EXPR_CALL)
202                 return get_func_constraint(expr);
203         if (expr->type == EXPR_BINOP)
204                 return expr_to_str(expr);
205         name = get_toplevel_name(expr);
206         if (name)
207                 return name;
208         return get_member_name(expr);
209 }

211 static int save_int_callback(void *_p, int argc, char **argv, char **azColName)
212 {
213         int *p = _p;

215         *p = atoi(argv[0]);
216         return 0;
217 }

219 static int constraint_str_to_id(const char *str)
220 {
221         int id = -1;

223         run_sql(save_int_callback, &id,
224                 "select id from constraints where str = '%q'", str);

226         return id;
227 }

229 static int save_constraint_str(void *_str, int argc, char **argv, char **azColNa
230 {
231         char **str = _str;

233         *str = alloc_string(argv[0]);
234         return 0;
235 }

237 static char *constraint_id_to_str(int id)
238 {
239         char *str = NULL;

241         run_sql(save_constraint_str, &str,
242                 "select str from constraints where id = '%d'", id);

244         return str;
245 }

247 static int save_op_callback(void *_p, int argc, char **argv, char **azColName)
248 {
249         int *p = _p;

251         if (argv[0][0] == '<' && argv[0][1] == '=')
252                 *p = SPECIAL_LTE;
253         else
254                 *p = '<';
255         return 0;
256 }

258 static int save_str_callback(void *_p, int argc, char **argv, char **azColName)
```

```
259 {
260         char **p = _p;
261 
262         if (!*p) {
263                 *p = alloc_string(argv[0]);
264         } else {
265                 char buf[256];
266 
267                 snprintf(buf, sizeof(buf), "%s, %s", *p, argv[0]);
268                 *p = alloc_string(buf);
269         }
270         return 0;
271 }
272 
273 char *get_required_constraint(const char *data_str)
274 {
275         char *required = NULL;
276 
277         run_sql(save_str_callback, &required,
278                 "select bound from constraints_required where data = '%q'", data
279 
280         return required;
281 }
282 
283 static int get_required_op(char *data_str, char *con_str)
284 {
285         int op = 0;
286 
287         run_sql(save_op_callback, &op,
288                 "select op from constraints_required where data = '%q' and bound
289 
290         return op;
291 }
292 
293 char *unmet_constraint(struct expression *data, struct expression *offset)
294 {
295         struct smatch_state *state;
296         struct constraint_list *list;
297         struct constraint *con;
298         char *data_str;
299         char *required;
300         int req_op;
301 
302         data_str = get_constraint_str(data);
303         if (!data_str)
304                 return NULL;
305 
306         required = get_required_constraint(data_str);
307         if (!required)
308                 goto free_data;
309 
310         state = get_state_expr(my_id, offset);
311         if (!state)
312                 goto free_data;
313         list = state->data;
314 
315         /* check the list of bounds on our index against the list that work */
316         FOR_EACH_PTR(list, con) {
317                 char *con_str;
318 
319                 con_str = constraint_id_to_str(con->id);
320                 if (!con_str) {
321                         sm_msg("constraint %d not found", con->id);
322                         continue;
323                 }
```



```
324 
325                 req_op = get_required_op(data_str, con_str);
326                 free_string(con_str);
327                 if (!req_op)
328                         continue;
329                 if (con->op == '<' || con->op == req_op) {
330                         free_string(required);
331                         required = NULL;
332                         goto free_data;
333                 }
334         } END_FOR_EACH_PTR(con);
335 
336 free_data:
337         free_string(data_str);
338         return required;
339 }
340 
341 struct string_list *saved_constraints;
342 static void save_new_constraint(const char *con)
343 {
344         if (list_has_string(saved_constraints, con))
345                 return;
346         insert_string(&saved_constraints, con);
347         sql_save_constraint(con);
348 }
349 
350 static void handle_comparison(struct expression *left, int op, struct expression
351 {
352         struct constraint_list *constraints;
353         struct smatch_state *state;
354         char *constraint;
355         int constraint_id;
356         int orig_op = op;
357         sval_t sval;
358 
359         /* known values are handled in smatch extra */
360         if (get_value(left, &sval) || get_value(right, &sval))
361                 return;
362 
363         if (local_debug)
364                 sm_msg("COMPARE: %s %s %s", expr_to_str(left), show_special(op),
365 
366         constraint = get_constraint_str(right);
367         if (!constraint)
368                 return;
369         if (local_debug)
370                 sm_msg("EXPR: %s CONSTRAINT %s", expr_to_str(right), constraint)
371         constraint_id = constraint_str_to_id(constraint);
372         if (local_debug)
373                 sm_msg("CONSTRAINT ID %d", constraint_id);
374         if (constraint_id < 0)
375                 save_new_constraint(constraint);
376         free_string(constraint);
377         if (constraint_id < 0)
378                 return;
379 
380         constraints = get_constraints(left);
381         constraints = clone_constraint_list(constraints);
382         op = negate_gt(orig_op);
383         add_constraint(&constraints, remove_unsigned_from_comparison(op), constr
384         state = alloc_constraint_state(constraints);
385 
386         if (op == orig_op) {
387                 if (local_debug)
388                         sm_msg("SETTING %s true %s", expr_to_str(left), state->n
389                 set_true_false_states_expr(my_id, left, state, NULL);
390         } else {
```

```
391                 if (local_debug)
392                         sm_msg("SETTING %s false %s", expr_to_str(left), state->
393
394                 set_true_false_states_expr(my_id, left, NULL, state);
395         }
396 }
397
398 static void match_condition(struct expression *expr)
399 {
400         if (expr->type != EXPR_COMPARE)
401                 return;
402
403         if (expr->op == SPECIAL_EQUAL ||
404             expr->op == SPECIAL_NOTEQUAL)
405                 return;
406
407         handle_comparison(expr->left, expr->op, expr->right);
408         handle_comparison(expr->right, flip_comparison(expr->op), expr->left);
409 }
410
411 struct constraint_list *get_constraints(struct expression *expr)
412 {
413         struct smatch_state *state;
414
415         state = get_state_expr(my_id, expr);
416         if (!state)
417                 return NULL;
418         return state->data;
419 }
420
421 static void match_caller_info(struct expression *expr)
422 {
423         struct expression *tmp;
424         struct smatch_state *state;
425         int i;
426
427         i = -1;
428         FOR_EACH_PTR(expr->args, tmp) {
429                 i++;
430                 state = get_state_expr(my_id, tmp);
431                 if (!state || state == &merged || state == &undefined)
432                         continue;
433                 sql_insert_caller_info(expr, CONSTRAINT, i, "$", state->name);
434         } END_FOR_EACH_PTR(tmp);
435 }
436
437 static void struct_member_callback(struct expression *call, int param, char *pri
438 {
439         if (sm->state == &merged || sm->state == &undefined)
440                 return;
441         sql_insert_caller_info(call, CONSTRAINT, param, printed_name, sm->state-
442 }
443
444 static struct smatch_state *constraint_str_to_state(char *value)
445 {
446         struct constraint_list *list = NULL;
447         char *p = value;
448         int op;
449         long long id;
450
451         while (true) {
452                 op = '<';
453                 if (*p != '<')
454                         return &undefined;
455                 p++;
456                 if (*p == '=') {
```



```
457                         op = SPECIAL_LTE;
458                         p++;
459                 }
460                 id = strtoll(p, &p, 10);
461                 add_constraint(&list, op, id);
462                 if (*p != ',')
463                         break;
464                 p++;
465                 if (*p != ' ')
466                         return &undefined;
467         }
468
469         return alloc_constraint_state(list);
470 }
471
472 static void set_param_constrained(const char *name, struct symbol *sym, char *ke
473 {
474         char fullname[256];
475
476         if (strcmp(key, "*$") == 0)
477                 snprintf(fullname, sizeof(fullname), "*%s", name);
478         else if (strncmp(key, "$", 1) == 0)
479                 snprintf(fullname, 256, "%s%s", name, key + 1);
480         else
481                 return;
482
483         set_state(my_id, name, sym, constraint_str_to_state(value));
484 }
485
486 static void print_return_implies_constrained(int return_id, char *return_ranges,
487 {
488         struct smatch_state *orig;
489         struct sm_state *sm;
490         const char *param_name;
491         int param;
492
493         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
494                 if (sm->state == &merged || sm->state == &undefined)
495                         continue;
496
497                 param = get_param_num_from_sym(sm->sym);
498                 if (param < 0)
499                         continue;
500
501                 orig = get_state_stree(get_start_states(), my_id, sm->name, sm->
502                 if (orig && strcmp(sm->state->name, orig->name) == 0)
503                         continue;
504
505                 param_name = get_param_name(sm);
506                 if (!param_name)
507                         continue;
508
509                 sql_insert_return_states(return_id, return_ranges, CONSTRAINT,
510                                          param, param_name, sm->state->name);
511         } END_FOR_EACH_SM(sm);
512 }
513
514 static void db_returns_constrained(struct expression *expr, int param, char *key
515 {
516         char *name;
517         struct symbol *sym;
518
519         name = return_state_to_var_sym(expr, param, key, &sym);
520         if (!name || !sym)
521                 goto free;
```

```
523         set_state(my_id, name, sym, constraint_str_to_state(value));
524 free:
525         free_string(name);
526 }

528 void register_constraints(int id)
529 {
530         my_id = id;

532         add_merge_hook(my_id, &merge_func);
533         add_hook(&match_condition, CONDITION_HOOK);

535         add_hook(&match_caller_info, FUNCTION_CALL_HOOK);
536         add_member_info_callback(my_id, struct_member_callback);
537         select_caller_info_hook(&set_param_constrained, CONSTRAINT);

539         add_split_return_callback(print_return_implies_constrained);
540         select_return_states_hook(CONSTRAINT, &db_returns_constrained);
541 }
```

```
**********************************************************
   12282 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_constraints_required.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;

struct allocator {
        const char *func;
        int param;
        int param2;
};

static struct allocator generic_allocator_table[] = {
        {"malloc", 0},
        {"memdup", 1},
        {"realloc", 1},
};

static struct allocator kernel_allocator_table[] = {
        {"kmalloc", 0},
        {"kzalloc", 0},
        {"vmalloc", 0},
        {"__vmalloc", 0},
        {"vzalloc", 0},
        {"sock_kmalloc", 1},
        {"kmemdup", 1},
        {"kmemdup_user", 1},
        {"dma_alloc_attrs", 1},
        {"pci_alloc_consistent", 1},
        {"pci_alloc_coherent", 1},
        {"devm_kmalloc", 1},
        {"devm_kzalloc", 1},
        {"krealloc", 1},
};

static struct allocator calloc_table[] = {
        {"calloc", 0, 1},
        {"kcalloc", 0, 1},
        {"kmalloc_array", 0, 1},
        {"devm_kcalloc", 1, 2},
};

static int bytes_per_element(struct expression *expr)
{
```



```
        struct symbol *type;

        type = get_type(expr);
        if (!type)
                return 0;

        if (type->type != SYM_PTR && type->type != SYM_ARRAY)
                return 0;

        type = get_base_type(type);
        return type_bytes(type);
}

static void save_constraint_required(struct expression *pointer, int op, struct
{
        char *data, *limit;

        data = get_constraint_str(pointer);
        if (!data)
                return;

        limit = get_constraint_str(constraint);
        if (!limit) {
                // FIXME deal with <= also
                if (op == '<')
                        set_state_expr(my_id, constraint, alloc_state_expr(point
                goto free_data;
        }

        sql_save_constraint_required(data, op, limit);

        free_string(limit);
free_data:
        free_string(data);
}

static int handle_zero_size_arrays(struct expression *pointer, struct expression
{
        struct expression *left, *right;
        struct symbol *type, *array, *array_type;
        sval_t struct_size;
        char *limit;
        char data[128];

        if (size->type != EXPR_BINOP || size->op != '+')
                return 0;

        type = get_type(pointer);
        if (!type || type->type != SYM_PTR)
                return 0;
        type = get_real_base_type(type);
        if (!type || !type->ident || type->type != SYM_STRUCT)
                return 0;
        if (!last_member_is_resizable(type))
                return 0;
        array = last_ptr_list((struct ptr_list *)type->symbol_list);
        if (!array || !array->ident)
                return 0;
        array_type = get_real_base_type(array);
        if (!array_type || array_type->type != SYM_ARRAY)
                return 0;
        array_type = get_real_base_type(array_type);

        left = strip_expr(size->left);
        right = strip_expr(size->right);
```

```
127         if (!get_implied_value(left, &struct_size))
128                 return 0;
129         if (struct_size.value != type_bytes(type))
130                 return 0;
131
132         if (right->type == EXPR_BINOP && right->op == '*') {
133                 struct expression *mult_left, *mult_right;
134                 sval_t sval;
135
136                 mult_left = strip_expr(right->left);
137                 mult_right = strip_expr(right->right);
138
139                 if (get_implied_value(mult_left, &sval) &&
140                     sval.value == type_bytes(array_type))
141                         size = mult_right;
142                 else if (get_implied_value(mult_right, &sval) &&
143                     sval.value == type_bytes(array_type))
144                         size = mult_left;
145                 else
146                         return 0;
147         }
148
149         snprintf(data, sizeof(data), "(struct %s)->%s", type->ident->name, array
150         limit = get_constraint_str(size);
151         if (!limit) {
152                 set_state_expr(my_id, size, alloc_state_expr(
153                                 member_expression(deref_expression(pointer), '*',
154                 return 1;
155         }
156
157         sql_save_constraint_required(data, '<', limit);
158
159         free_string(limit);
160         return 1;
161 }
162
163 static void match_alloc_helper(struct expression *pointer, struct expression *si
164 {
165         struct expression *size_orig, *tmp;
166         sval_t sval;
167         int cnt = 0;
168
169         pointer = strip_expr(pointer);
170         size = strip_expr(size);
171         if (!size || !pointer)
172                 return;
173
174         size_orig = size;
175         if (recurse) {
176                 while ((tmp = get_assigned_expr(size))) {
177                         size = strip_expr(tmp);
178                         if (cnt++ > 5)
179                                 break;
180                 }
181                 if (size != size_orig) {
182                         match_alloc_helper(pointer, size, 0);
183                         size = size_orig;
184                 }
185         }
186
187         if (handle_zero_size_arrays(pointer, size))
188                 return;
189
190         if (size->type == EXPR_BINOP && size->op == '*') {
191                 struct expression *mult_left, *mult_right;
```



```
193                 mult_left = strip_expr(size->left);
194                 mult_right = strip_expr(size->right);
195
196                 if (get_implied_value(mult_left, &sval) &&
197                     sval.value == bytes_per_element(pointer))
198                         size = mult_right;
199                 else if (get_implied_value(mult_right, &sval) &&
200                     sval.value == bytes_per_element(pointer))
201                         size = mult_left;
202                 else
203                         return;
204         }
205
206         if (size->type == EXPR_BINOP && size->op == '+' &&
207             get_implied_value(size->right, &sval) &&
208             sval.value == 1)
209                 save_constraint_required(pointer, SPECIAL_LTE, size->left);
210         else
211                 save_constraint_required(pointer, '<', size);
212 }
213
214 static void match_alloc(const char *fn, struct expression *expr, void *_size_arg
215 {
216         int size_arg = PTR_INT(_size_arg);
217         struct expression *call, *arg;
218
219         call = strip_expr(expr->right);
220         arg = get_argument_from_call_expr(call->args, size_arg);
221
222         match_alloc_helper(expr->left, arg, 1);
223 }
224
225 static void match_calloc(const char *fn, struct expression *expr, void *_start_a
226 {
227         struct expression *pointer, *call, *size;
228         struct expression *count = NULL;
229         int start_arg = PTR_INT(_start_arg);
230         sval_t sval;
231
232         pointer = strip_expr(expr->left);
233         call = strip_expr(expr->right);
234
235         size = get_argument_from_call_expr(call->args, start_arg);
236         if (get_implied_value(size, &sval) &&
237             sval.value == bytes_per_element(pointer))
238                 count = get_argument_from_call_expr(call->args, start_arg + 1);
239         else {
240                 size = get_argument_from_call_expr(call->args, start_arg + 1);
241                 if (get_implied_value(size, &sval) &&
242                     sval.value == bytes_per_element(pointer))
243                         count = get_argument_from_call_expr(call->args, start_ar
244         }
245
246         if (!count)
247                 return;
248
249         save_constraint_required(pointer, '<', count);
250 }
251
252 static void add_allocation_function(const char *func, void *call_back, int param
253 {
254         add_function_assign_hook(func, call_back, INT_PTR(param));
255 }
256
257 static void match_assign_size(struct expression *expr)
258 {
```

```
259         struct smatch_state *state;
260         char *data, *limit;

262         state = get_state_expr(my_id, expr->right);
263         if (!state || !state->data)
264                 return;

266         data = get_constraint_str(state->data);
267         if (!data)
268                 return;

270         limit = get_constraint_str(expr->left);
271         if (!limit)
272                 goto free_data;

274         sql_save_constraint_required(data, '<', limit);

276         free_string(limit);
277 free_data:
278         free_string(data);
279 }

281 static void match_assign_has_buf_comparison(struct expression *expr)
282 {
283         struct expression *size;

285         if (expr->op != '=')
286                 return;
287         if (expr->right->type == EXPR_CALL)
288                 return;
289         size = get_size_variable(expr->right);
290         if (!size)
291                 return;
292         match_alloc_helper(expr->left, size, 1);
293 }

295 static void match_assign_data(struct expression *expr)
296 {
297         struct expression *right, *arg, *tmp;
298         int i;
299         int size_arg;
300         int size_arg2 = -1;

302         if (expr->op != '=')
303                 return;

305         /* Direct calls are handled else where (for now at least) */
306         tmp = get_assigned_expr(expr->right);
307         if (!tmp)
308                 return;

310         right = strip_expr(tmp);
311         if (right->type != EXPR_CALL)
312                 return;

314         if (right->fn->type != EXPR_SYMBOL ||
315             !right->fn->symbol ||
316             !right->fn->symbol->ident)
317                 return;

319         for (i = 0; i < ARRAY_SIZE(generic_allocator_table); i++) {
320                 if (strcmp(right->fn->symbol->ident->name,
321                            generic_allocator_table[i].func) == 0) {
322                         size_arg = generic_allocator_table[i].param;
323                         goto found;
324                 }
```



```
325         }

327         if (option_project != PROJ_KERNEL)
328                 return;

330         for (i = 0; i < ARRAY_SIZE(kernel_allocator_table); i++) {
331                 if (strcmp(right->fn->symbol->ident->name,
332                            kernel_allocator_table[i].func) == 0) {
333                         size_arg = kernel_allocator_table[i].param;
334                         goto found;
335                 }
336         }

338         for (i = 0; i < ARRAY_SIZE(calloc_table); i++) {
339                 if (strcmp(right->fn->symbol->ident->name,
340                            calloc_table[i].func) == 0) {
341                         size_arg = calloc_table[i].param;
342                         size_arg2 = calloc_table[i].param2;
343                         goto found;
344                 }
345         }

347         return;

349 found:
350         arg = get_argument_from_call_expr(right->args, size_arg);
351         match_alloc_helper(expr->left, arg, 1);
352         if (size_arg2 == -1)
353                 return;
354         arg = get_argument_from_call_expr(right->args, size_arg2);
355         match_alloc_helper(expr->left, arg, 1);
356 }

358 static void match_assign_ARRAY_SIZE(struct expression *expr)
359 {
360         struct expression *array;
361         char *data, *limit;
362         const char *macro;

364         macro = get_macro_name(expr->right->pos);
365         if (!macro || strcmp(macro, "ARRAY_SIZE") != 0)
366                 return;
367         array = strip_expr(expr->right);
368         if (array->type != EXPR_BINOP || array->op != '+')
369                 return;
370         array = strip_expr(array->left);
371         if (array->type != EXPR_BINOP || array->op != '/')
372                 return;
373         array = strip_expr(array->left);
374         if (array->type != EXPR_SIZEOF)
375                 return;
376         array = strip_expr(array->cast_expression);
377         if (array->type != EXPR_PREOP || array->op != '*')
378                 return;
379         array = strip_expr(array->unop);

381         data = get_constraint_str(array);
382         limit = get_constraint_str(expr->left);
383         if (!data || !limit)
384                 goto free;

386         sql_save_constraint_required(data, '<', limit);

388 free:
389         free_string(data);
390         free_string(limit);
```

```
391 }
393 static void match_assign_buf_comparison(struct expression *expr)
394 {
395         struct expression *pointer;
397         if (expr->op != '=')
398                 return;
399         pointer = get_array_variable(expr->right);
400         if (!pointer)
401                 return;
403         match_alloc_helper(pointer, expr->right, 1);
404 }
406 static int constraint_found(void *_found, int argc, char **argv, char **azColNam
407 {
408         int *found = _found;
410         *found = 1;
411         return 0;
412 }
414 static int has_constraint(struct expression *expr, const char *constraint)
415 {
416         int found = 0;
418         if (get_state_expr(my_id, expr))
419                 return 1;
421         run_sql(constraint_found, &found,
422                 "select data from constraints_required where bound = '%q' limit
423                 escape_newlines(constraint));
425         return found;
426 }
428 static void match_assign_constraint(struct expression *expr)
429 {
430         struct symbol *type;
431         char *left, *right;
433         if (expr->op != '=')
434                 return;
436         type = get_type(expr->left);
437         if (!type || type->type != SYM_BASETYPE)
438                 return;
440         left = get_constraint_str(expr->left);
441         if (!left)
442                 return;
443         right = get_constraint_str(expr->right);
444         if (!right)
445                 goto free;
446         if (!has_constraint(expr->right, right))
447                 return;
448         sql_copy_constraint_required(left, right);
449 free:
450         free_string(right);
451         free_string(left);
452 }
454 void register_constraints_required(int id)
455 {
456         my_id = id;
```



```
458         add_hook(&match_assign_size, ASSIGNMENT_HOOK);
459         add_hook(&match_assign_data, ASSIGNMENT_HOOK);
460         add_hook(&match_assign_has_buf_comparison, ASSIGNMENT_HOOK);
462         add_hook(&match_assign_ARRAY_SIZE, ASSIGNMENT_HOOK);
463         add_hook(&match_assign_ARRAY_SIZE, GLOBAL_ASSIGNMENT_HOOK);
464         add_hook(&match_assign_buf_comparison, ASSIGNMENT_HOOK);
465         add_hook(&match_assign_constraint, ASSIGNMENT_HOOK);
467         add_allocation_function("malloc", &match_alloc, 0);
468         add_allocation_function("memdup", &match_alloc, 1);
469         add_allocation_function("realloc", &match_alloc, 1);
470         add_allocation_function("realloc", &match_calloc, 0);
471         if (option_project == PROJ_KERNEL) {
472                 add_allocation_function("kmalloc", &match_alloc, 0);
473                 add_allocation_function("kzalloc", &match_alloc, 0);
474                 add_allocation_function("vmalloc", &match_alloc, 0);
475                 add_allocation_function("__vmalloc", &match_alloc, 0);
476                 add_allocation_function("vzalloc", &match_alloc, 0);
477                 add_allocation_function("sock_kmalloc", &match_alloc, 1);
478                 add_allocation_function("kmemdup", &match_alloc, 1);
479                 add_allocation_function("kmemdup_user", &match_alloc, 1);
480                 add_allocation_function("dma_alloc_attrs", &match_alloc, 1);
481                 add_allocation_function("pci_alloc_consistent", &match_alloc, 1)
482                 add_allocation_function("pci_alloc_coherent", &match_alloc, 1);
483                 add_allocation_function("devm_kmalloc", &match_alloc, 1);
484                 add_allocation_function("devm_kzalloc", &match_alloc, 1);
485                 add_allocation_function("kcalloc", &match_calloc, 0);
486                 add_allocation_function("kmalloc_array", &match_calloc, 0);
487                 add_allocation_function("devm_kcalloc", &match_calloc, 1);
488                 add_allocation_function("krealloc", &match_alloc, 1);
489         }
490 }
```

```
**********************************************************
   14686 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_container_of.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;
static int param_id;

static struct stree *used_stree;
static struct stree_stack *saved_stack;

STATE(used);

int get_param_from_container_of(struct expression *expr)
{
        struct expression *param_expr;
        struct symbol *type;
        sval_t sval;
        int param;


        type = get_type(expr);
        if (!type || type->type != SYM_PTR)
                return -1;

        expr = strip_expr(expr);
        if (expr->type != EXPR_BINOP || expr->op != '-')
                return -1;

        if (!get_value(expr->right, &sval))
                return -1;
        if (sval.value < 0 || sval.value > 4096)
                return -1;

        param_expr = get_assigned_expr(expr->left);
        if (!param_expr)
                return -1;
        param = get_param_num(param_expr);
        if (param < 0)
                return -1;

        return param;
}
```



```
int get_offset_from_container_of(struct expression *expr)
{
        struct expression *param_expr;
        struct symbol *type;
        sval_t sval;

        type = get_type(expr);
        if (!type || type->type != SYM_PTR)
                return -1;

        expr = strip_expr(expr);
        if (expr->type != EXPR_BINOP || expr->op != '-')
                return -1;

        if (!get_value(expr->right, &sval))
                return -1;
        if (sval.value < 0 || sval.value > 4096)
                return -1;

        param_expr = get_assigned_expr(expr->left);
        if (!param_expr)
                return -1;

        return sval.value;
}

static int get_container_arg(struct symbol *sym)
{
        struct expression *__mptr;
        int param;

        if (!sym || !sym->ident)
                return -1;

        __mptr = get_assigned_expr_name_sym(sym->ident->name, sym);
        param = get_param_from_container_of(__mptr);

        return param;
}

static int get_container_offset(struct symbol *sym)
{
        struct expression *__mptr;
        int offset;

        if (!sym || !sym->ident)
                return -1;

        __mptr = get_assigned_expr_name_sym(sym->ident->name, sym);
        offset = get_offset_from_container_of(__mptr);

        return offset;
}

static char *get_container_name(struct sm_state *sm, int offset)
{
        static char buf[256];
        const char *name;

        name = get_param_name(sm);
        if (!name)
                return NULL;

        if (name[0] == '$')
                snprintf(buf, sizeof(buf), "$(-%d)%s", offset, name + 1);
        else if (name[0] == '*' || name[1] == '$')
```

```
127                 snprintf(buf, sizeof(buf), "*$(-%d)%s", offset, name + 2);
128         else
129                 return NULL;
130
131         return buf;
132 }
133
134 static void get_state_hook(int owner, const char *name, struct symbol *sym)
135 {
136         int arg;
137
138         if (!option_info)
139                 return;
140         if (__in_fake_assign)
141                 return;
142
143         arg = get_container_arg(sym);
144         if (arg >= 0)
145                 set_state_stree(&used_stree, my_id, name, sym, &used);
146 }
147
148 static void set_param_used(struct expression *call, struct expression *arg, char
149 {
150         struct symbol *sym;
151         char *name;
152         int arg_nr;
153
154         name = get_variable_from_key(arg, key, &sym);
155         if (!name || !sym)
156                 goto free;
157
158         arg_nr = get_container_arg(sym);
159         if (arg_nr >= 0)
160                 set_state(my_id, name, sym, &used);
161 free:
162         free_string(name);
163 }
164
165 static void process_states(void)
166 {
167         struct sm_state *tmp;
168         int arg, offset;
169         const char *name;
170
171         FOR_EACH_SM(used_stree, tmp) {
172                 arg = get_container_arg(tmp->sym);
173                 offset = get_container_offset(tmp->sym);
174                 if (arg < 0 || offset < 0)
175                         continue;
176                 name = get_container_name(tmp, offset);
177                 if (!name)
178                         continue;
179                 sql_insert_return_implies(CONTAINER, arg, name, "");
180         } END_FOR_EACH_SM(tmp);
181
182         free_stree(&used_stree);
183 }
184
185 static void match_function_def(struct symbol *sym)
186 {
187         free_stree(&used_stree);
188 }
189
190 static void match_save_states(struct expression *expr)
191 {
192         push_stree(&saved_stack, used_stree);
```



```
193         used_stree = NULL;
194 }
195
196 static void match_restore_states(struct expression *expr)
197 {
198         free_stree(&used_stree);
199         used_stree = pop_stree(&saved_stack);
200 }
201
202 static void print_returns_container_of(int return_id, char *return_ranges, struc
203 {
204         int offset;
205         int param;
206         char key[64];
207         char value[64];
208
209         param = get_param_from_container_of(expr);
210         if (param < 0)
211                 return;
212         offset = get_offset_from_container_of(expr);
213         if (offset < 0)
214                 return;
215
216         snprintf(key, sizeof(key), "%d", param);
217         snprintf(value, sizeof(value), "-%d", offset);
218
219         /* no need to add it to return_implies because it's not really param_use
220         sql_insert_return_states(return_id, return_ranges, CONTAINER, -1,
221                                  key, value);
222 }
223
224 static void returns_container_of(struct expression *expr, int param, char *key,
225 {
226         struct expression *call, *arg;
227         int offset;
228         char buf[64];
229
230         if (expr->type != EXPR_ASSIGNMENT || expr->op != '=')
231                 return;
232         call = strip_expr(expr->right);
233         if (call->type != EXPR_CALL)
234                 return;
235         if (param != -1)
236                 return;
237         param = atoi(key);
238         offset = atoi(value);
239
240         arg = get_argument_from_call_expr(call->args, param);
241         if (!arg)
242                 return;
243         if (arg->type != EXPR_SYMBOL)
244                 return;
245         param = get_param_num(arg);
246         if (param < 0)
247                 return;
248         snprintf(buf, sizeof(buf), "$(%d)", offset);
249         sql_insert_return_implies(CONTAINER, param, buf, "");
250 }
251
252 static int get_shared_cnt(const char *one, const char *two)
253 {
254         int i;
255         int on_end = false;
256
257         i = 0;
258         while (true) {
```

```
259                 if (!one[i] || !two[i]) {
260                         on_end = true;
261                         break;
262                 }
263                 if (one[i] != two[i])
264                         break;
265                 i++;
266         }
267         if (i == 0)
268                 return 0;
269         i--;
270         while (i > 0 && (one[i] == '>' || one[i] == '-' || one[i] == '.')) {
271                 on_end = true;
272                 i--;
273         }
274         if (!on_end)
275                 return 0;

277         return i + 1;
278 }

280 static int build_offset_str(struct expression *expr, const char *name,
281                             int shared, char *buf, int size, int op)
282 {
283         int chop = 0;
284         int offset;
285         int i;

287         i = shared;
288         while (name[i]) {
289                 if (name[i] == '.' || name[i] == '-')
290                         chop++;
291                 i++;
292         }

294         // FIXME:  Handle more chops
295         if (chop > 1)
296                 return 0;

298         if (chop == 0) {
299                 offset = 0;
300         } else {
301                 offset = get_member_offset_from_deref(expr);
302                 if (offset < 0)
303                         return 0;
304         }

306         snprintf(buf, size, "%c%d", (op == '+') ? '+' : '-', offset);
307         return 1;
308 }

310 static void match_call(struct expression *call)
311 {
312         struct expression *fn, *arg;
313         char *fn_name, *arg_name;
314         int param, shared;
315         char minus_str[64];
316         char plus_str[64];
317         char offset_str[64];
318         bool star;

320         /*
321          * We're trying to link the function with the parameter.  There are a
322          * couple ways this can be passed:
323          * foo->func(foo, ...);
324          * foo->func(foo->x, ...);
```



```
325          * foo->bar.func(&foo->bar, ...);
326          * foo->bar->baz->func(foo, ...);
327          *
328          * So the method is basically to subtract the offsets until we get to
329          * the common bit, then add the member offsets to get the parameter.
330          *
331          * If we're taking an address then the offset math is not stared,
332          * otherwise it is.  Starred means dereferenced.
333          */
334         fn = strip_expr(call->fn);
335         fn_name = expr_to_var(fn);
336         if (!fn_name)
337                 return;

339         param = -1;
340         FOR_EACH_PTR(call->args, arg) {
341                 param++;

343                 arg = strip_expr(arg);
344                 star = true;
345                 if (arg->type == EXPR_PREOP && arg->op == '&') {
346                         arg = strip_expr(arg->unop);
347                         star = false;
348                 }

350                 arg_name = expr_to_var(arg);
351                 if (!arg_name)
352                         continue;
353                 shared = get_shared_cnt(fn_name, arg_name);
354                 if (!shared)
355                         goto free_arg_name;
356                 if (!build_offset_str(fn, fn_name, shared, minus_str, sizeof(min
357                         goto free_arg_name;
358                 if (!build_offset_str(arg, arg_name, shared, plus_str, sizeof(pl
359                         goto free_arg_name;
360                 if (star)
361                         snprintf(offset_str, sizeof(offset_str), "*(%s%s)", minu
362                 else
363                         snprintf(offset_str, sizeof(offset_str), "%s%s", minus_s
364                 sql_insert_caller_info(call, CONTAINER, param, offset_str, "$(-1
365 free_arg_name:
366                 free_string(arg_name);
367         } END_FOR_EACH_PTR(arg);

369         free_string(fn_name);
370 }

372 static void db_passed_container(const char *name, struct symbol *sym, char *key,
373 {
374         sval_t offset = {
375                 .type = &int_ctype,
376         };
377         const char *arg_offset;
378         int star = 0;
379         int val;

381         if (key[0] == '*') {
382                 star = 1;
383                 key += 2;
384         }

386         val = atoi(key);
387         if (val < -4095 || val > 0)
388                 return;
389         offset.value = -val;
390         arg_offset = strchr(key, '+');
```

```
391         if (!arg_offset)
392                 return;
393         val = atoi(arg_offset + 1);
394         if (val > 4095 || val < 0)
395                 return;
396         offset.value |= val << 16;
397         if (star)
398                 offset.value |= 1ULL << 31;

400         set_state(param_id, name, sym, alloc_estate_sval(offset));
401 }

403 struct db_info {
404         struct symbol *arg;
405         int prev_offset;
406         struct range_list *rl;
407         int star;
408         struct stree *stree;
409 };

411 static struct symbol *get_member_from_offset(struct symbol *sym, int offset)
412 {
413         struct symbol *type, *tmp;
414         int cur;

416         type = get_real_base_type(sym);
417         if (!type || type->type != SYM_PTR)
418                 return NULL;
419         type = get_real_base_type(type);
420         if (!type || type->type != SYM_STRUCT)
421                 return NULL;

423         cur = 0;
424         FOR_EACH_PTR(type->symbol_list, tmp) {
425                 cur = ALIGN(cur, tmp->ctype.alignment);
426                 if (offset == cur)
427                         return tmp;
428                 cur += type_bytes(tmp);
429         } END_FOR_EACH_PTR(tmp);
430         return NULL;
431 }

433 static struct symbol *get_member_type_from_offset(struct symbol *sym, int offset
434 {
435         struct symbol *base_type;
436         struct symbol *member;

438         base_type = get_real_base_type(sym);
439         if (base_type && base_type->type == SYM_PTR)
440                 base_type = get_real_base_type(base_type);
441         if (offset == 0 && base_type && base_type->type == SYM_BASETYPE)
442                 return base_type;

444         member = get_member_from_offset(sym, offset);
445         if (!member)
446                 return NULL;
447         return get_real_base_type(member);
448 }

450 static const char *get_name_from_offset(struct symbol *arg, int offset)
451 {
452         struct symbol *member, *type;
453         const char *name;
454         static char fullname[256];

456         name = arg->ident->name;
```



```
458         type = get_real_base_type(arg);
459         if (!type || type->type != SYM_PTR)
460                 return name;

462         type = get_real_base_type(type);
463         if (!type)
464                 return NULL;
465         if (type->type != SYM_STRUCT) {
466                 snprintf(fullname, sizeof(fullname), "*%s", name);
467                 return fullname;
468         }

470         member = get_member_from_offset(arg, offset);
471         if (!member)
472                 return NULL;

474         snprintf(fullname, sizeof(fullname), "%s->%s", name, member->ident->name
475         return fullname;
476 }

478 static void set_param_value(struct stree **stree, struct symbol *arg, int offset
479 {
480         const char *name;

482         name = get_name_from_offset(arg, offset);
483         if (!name)
484                 return;
485         set_state_stree(stree, SMATCH_EXTRA, name, arg, alloc_estate_rl(rl));
486 }

488 static int save_vals(void *_db_info, int argc, char **argv, char **azColName)
489 {
490         struct db_info *db_info = _db_info;
491         struct symbol *type;
492         struct range_list *rl;
493         int offset = 0;
494         const char *value;

496         if (argc == 2) {
497                 offset = atoi(argv[0]);
498                 value = argv[1];
499         } else {
500                 value = argv[0];
501         }

503         if (db_info->prev_offset != -1 &&
504             db_info->prev_offset != offset) {
505                 set_param_value(&db_info->stree, db_info->arg, db_info->prev_off
506                 db_info->rl = NULL;
507         }

509         db_info->prev_offset = offset;

511         type = get_real_base_type(db_info->arg);
512         if (db_info->star)
513                 goto found_type;
514         if (type->type != SYM_PTR)
515                 return 0;
516         type = get_real_base_type(type);
517         if (type->type == SYM_BASETYPE)
518                 goto found_type;
519         type = get_member_type_from_offset(db_info->arg, offset);
520 found_type:
521         str_to_rl(type, (char *)value, &rl);
522         if (db_info->rl)
```

```
523                 db_info->rl = rl_union(db_info->rl, rl);
524         else
525                 db_info->rl = rl;
526
527         return 0;
528 }
529
530 static struct stree *load_tag_info_sym(mtag_t tag, struct symbol *arg, int arg_o
531 {
532         struct db_info db_info = {
533                 .arg = arg,
534                 .prev_offset = -1,
535                 .star = star,
536         };
537         struct symbol *type;
538
539         if (!tag || !arg->ident)
540                 return NULL;
541
542         type = get_real_base_type(arg);
543         if (!type)
544                 return NULL;
545         if (!star) {
546                 if (type->type != SYM_PTR)
547                         return NULL;
548                 type = get_real_base_type(type);
549                 if (!type)
550                         return NULL;
551         }
552
553         if (star || type->type == SYM_BASETYPE) {
554                 run_sql(save_vals, &db_info,
555                         "select value from mtag_data where tag = %lld and offset
556                         tag, arg_offset, DATA_VALUE);
557         } else {  /* presumably the parameter is a struct pointer */
558                 run_sql(save_vals, &db_info,
559                         "select offset, value from mtag_data where tag = %lld an
560                         tag, DATA_VALUE);
561         }
562
563         if (db_info.prev_offset != -1)
564                 set_param_value(&db_info.stree, arg, db_info.prev_offset, db_inf
565
566         // FIXME: handle an offset correctly
567         if (!star && !arg_offset) {
568                 sval_t sval;
569
570                 sval.type = get_real_base_type(arg);
571                 sval.uvalue = tag;
572                 set_state_stree(&db_info.stree, SMATCH_EXTRA, arg->ident->name,
573         }
574         return db_info.stree;
575 }
576
577 static void handle_passed_container(struct symbol *sym)
578 {
579         struct symbol *arg;
580         struct smatch_state *state;
581         struct sm_state *sm;
582         struct stree *stree;
583         mtag_t fn_tag, container_tag, arg_tag;
584         sval_t offset;
585         int container_offset, arg_offset;
586         int star;
587
588         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
```



```
589                 state = get_state(param_id, arg->ident->name, arg);
590                 if (state)
591                         goto found;
592         } END_FOR_EACH_PTR(arg);
593
594         return;
595 found:
596         if (!estate_get_single_value(state, &offset))
597                 return;
598         container_offset = -(offset.value & 0xffff);
599         arg_offset = (offset.value & 0xfff0000) >> 16;
600         star = !!(offset.value & (1ULL << 31));
601
602         if (!get_toplevel_mtag(cur_func_sym, &fn_tag))
603                 return;
604         if (!mtag_map_select_container(fn_tag, container_offset, &container_tag)
605                 return;
606         if (!arg_offset || star) {
607                 arg_tag = container_tag;
608         } else {
609                 if (!mtag_map_select_tag(container_tag, -arg_offset, &arg_tag))
610                         return;
611         }
612
613         stree = load_tag_info_sym(arg_tag, arg, arg_offset, star);
614         FOR_EACH_SM(stree, sm) {
615                 set_state(sm->owner, sm->name, sm->sym, sm->state);
616         } END_FOR_EACH_SM(sm);
617         free_stree(&stree);
618 }
619
620 void register_container_of(int id)
621 {
622         my_id = id;
623
624         add_hook(&match_function_def, FUNC_DEF_HOOK);
625
626         add_get_state_hook(&get_state_hook);
627
628         add_hook(&match_save_states, INLINE_FN_START);
629         add_hook(&match_restore_states, INLINE_FN_END);
630
631         select_return_implies_hook(CONTAINER, &set_param_used);
632         all_return_states_hook(&process_states);
633
634         add_split_return_callback(&print_returns_container_of);
635         select_return_states_hook(CONTAINER, &returns_container_of);
636
637         add_hook(&match_call, FUNCTION_CALL_HOOK);
638 }
639
640 static struct smatch_state *unmatched_state(struct sm_state *sm)
641 {
642         return alloc_estate_whole(estate_type(sm->state));
643 }
644
645 void register_container_of2(int id)
646 {
647         param_id = id;
648
649         select_caller_info_hook(db_passed_container, CONTAINER);
650         add_hook(&handle_passed_container, AFTER_DEF_HOOK);
651         add_unmatched_state_hook(param_id, &unmatched_state);
652         add_merge_hook(param_id, &merge_estates);
653 }
```

```
**********************************************************
    2095 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_data/db/build_early_index.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

db_file=$1


cat << EOF | sqlite3 $db_file
PRAGMA synchronous = OFF;
PRAGMA cache_size = 800000;
PRAGMA journal_mode = OFF;
PRAGMA count_changes = OFF;
PRAGMA temp_store = MEMORY;
PRAGMA locking = EXCLUSIVE;

CREATE INDEX caller_fn_idx on caller_info (function, call_id);
CREATE INDEX caller_ff_idx on caller_info (file, function, call_id);
CREATE INDEX common_fn_idx on common_caller_info (function, call_id);
CREATE INDEX common_ff_idx on common_caller_info (file, function, call_id);
CREATE INDEX call_implies_fn_idx on call_implies (function);
CREATE INDEX call_implies_ff_idx on call_implies (file, function);
CREATE INDEX return_implies_fn_idx on return_implies (function);
CREATE INDEX return_implies_ff_idx on return_implies (file, function);
CREATE INDEX data_file_info_idx on data_info (file, data);
CREATE INDEX data_info_idx on data_info (data);
CREATE INDEX fn_ptr_idx_file on function_ptr (file, function);
CREATE INDEX fn_ptr_idx_nofile on function_ptr (function);
CREATE INDEX fn_ptr_idx_ptr on function_ptr (ptr);
CREATE INDEX file_function_type_idx on function_type (file, function);
CREATE INDEX function_type_idx on function_type (function);
CREATE INDEX function_type_size_idx ON function_type_size (type);
CREATE INDEX function_type_value_idx ON function_type_value (type);
CREATE INDEX local_value_idx on local_values (file, variable);
CREATE INDEX return_states_fn_idx on return_states (function);
CREATE INDEX return_states_ff_idx on return_states (file, function);
CREATE INDEX parameter_name_file_idx on parameter_name (file, function);
CREATE INDEX parameter_name_idx on parameter_name (function);
CREATE INDEX str_idx on constraints (str);
CREATE INDEX required_idx on constraints_required (data);
CREATE INDEX mtag_about_idx on mtag_about (tag);
CREATE INDEX mtag_data_idx on mtag_data (tag);
CREATE INDEX mtag_map_idx1 on mtag_map (tag);
CREATE INDEX mtag_map_idx2 on mtag_map (container);
CREATE INDEX sink_index on sink_info (file, sink_name);

EOF

#CREATE INDEX type_size_idx on type_size (type);
#CREATE INDEX type_val_idx on type_value (type);
```

**********
*325 Fri Dec 21 15:00:16 2018*
*new/usr/src/tools/smatch/src/smatch_data/db/build_late_index.sh*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
#!/bin/bash

db_file=$1


cat << EOF | sqlite3 $db_file
PRAGMA synchronous = OFF;
PRAGMA cache_size = 800000;
PRAGMA journal_mode = OFF;
PRAGMA count_changes = OFF;
PRAGMA temp_store = MEMORY;
PRAGMA locking = EXCLUSIVE;

CREATE INDEX type_size_idx on type_size (type);
CREATE INDEX type_val_idx on type_value (type);

EOF
```

```
**********************************************************
     279 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_data/db/call_implies.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
CREATE TABLE call_implies (
        file varchar(128),
        function varchar(64),
        call_id integer,
        static boolean,
        type integer,
        parameter integer,
        key varchar(256),
        value varchar(256),

        CONSTRAINT implies_row UNIQUE (file, function, call_id, static, type, pa
);
```

```
**********************************************************
     192 Fri Dec 21 15:00:16 2018
new/usr/src/tools/smatch/src/smatch_data/db/caller_info.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE caller_info (file varchar(128), caller varchar(64), function varcha
```

```
**********************************************************
     137 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/clear_user_data.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #!/bin/bash

   3 echo "delete from caller_info where type = 8017; delete from return_states where
```

```
**********************************************************
     199 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/common_caller_info.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE common_caller_info (file varchar(128), caller varchar(64), function
```

```
**********************************************************
     116 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/constraints.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 CREATE TABLE constraints (
2         id integer primary key,
3         str varchar(256),
4
5         CONSTRAINT constraint_name UNIQUE (str)
6 );
```

```
**********************************************************
     142 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/constraints_required.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
CREATE TABLE constraints_required (
        data varchar(256),
        op integer,
        bound varchar(256),

        CONSTRAINT unique_row UNIQUE (data, op, bound)
);
```

```
**********************************************************
     879 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/copy_required_constraints.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use warnings;
use bigint;
use DBI;
use Data::Dumper;
use File::Basename;
use Try::Tiny;

my $project = shift;
$project =~ s/.*=(.*)/$1/;
my $warns = shift;
my $db_file = shift;

my $db;

sub connect_to_db($)
{
    my $name = shift;

    $db = DBI->connect("dbi:SQLite:$name", "", "", {AutoCommit => 0});

    $db->do("PRAGMA cache_size = 800000");
    $db->do("PRAGMA journal_mode = OFF");
    $db->do("PRAGMA count_changes = OFF");
    $db->do("PRAGMA temp_store = MEMORY");
    $db->do("PRAGMA locking = EXCLUSIVE");
}

sub copy_constraints($$)
{
    my $full_path = shift;
    my $project = shift;
    my $dir = dirname($full_path);

    $db->do('insert or ignore into constraints (str) select bound from constrain

    $db->commit();
}

connect_to_db($db_file);
copy_constraints($0, $project);

$db->commit();
$db->disconnect();
```

**********
1781 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/create_db.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#!/bin/bash

if echo $1 | grep -q '^-p' ; then
    PROJ=$(echo $1 | cut -d = -f 2)
    shift
fi

info_file=$1

if [[ "$info_file" = "" ]] ; then
    echo "Usage:  $0 -p=<project> <file with smatch messages>"
    exit 1
fi

bin_dir=$(dirname $0)
db_file=smatch_db.sqlite.new

rm -f $db_file

for i in ${bin_dir}/*.schema ; do
    cat $i | sqlite3 $db_file
done

${bin_dir}/init_constraints.pl "$PROJ" $info_file $db_file
${bin_dir}/init_constraints_required.pl "$PROJ" $info_file $db_file
${bin_dir}/fill_db_sql.pl "$PROJ" $info_file $db_file
if [ -e ${info_file}.sql ] ; then
    ${bin_dir}/fill_db_sql.pl "$PROJ" ${info_file}.sql $db_file
fi
${bin_dir}/fill_db_caller_info.pl "$PROJ" $info_file $db_file
if [ -e ${info_file}.caller_info ] ; then
    ${bin_dir}/fill_db_caller_info.pl "$PROJ" ${info_file}.caller_info $db_file
fi
${bin_dir}/build_early_index.sh $db_file

${bin_dir}/fill_db_type_value.pl "$PROJ" $info_file $db_file
${bin_dir}/fill_db_type_size.pl "$PROJ" $info_file $db_file
${bin_dir}/copy_required_constraints.pl "$PROJ" $info_file $db_file
${bin_dir}/build_late_index.sh $db_file

${bin_dir}/fixup_all.sh $db_file
if [ "$PROJ" != "" ] ; then
    ${bin_dir}/fixup_${PROJ}.sh $db_file
fi

${bin_dir}/remove_mixed_up_pointer_params.pl $db_file
${bin_dir}/mark_function_ptrs_searchable.pl $db_file

# delete duplicate entrees and speed things up
echo "delete from function_ptr where rowid not in (select min(rowid) from functi

test -e  ${bin_dir}/${PROJ}.return_fixes && \
cat ${bin_dir}/${PROJ}.return_fixes | \
while read func old new ; do
    echo "update return_states set return = '$new' where function = '$func' and
done

mv $db_file smatch_db.sqlite
```

```
**********************************************************
      94 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/data_info.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE data_info (file varchar(80), data varchar(80), type integer, value
```

```
**********************************************************
      36 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/db.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 PRAGMA max_page_count = 2147483646;
```

```
**********************************************************
    2267 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/fill_db_caller_info.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use DBI;
use Scalar::Util qw(looks_like_number);

sub usage()
{
    print "usage:  $0 <project> <smatch_warns.txt> <db_file>\n";
    exit(1);
}

my %too_common_funcs;
sub get_too_common_functions($$$)
{
    my $path = shift;
    my $project = shift;
    my $warns = shift;

    open(FUNCS, "grep 'SQL_caller_info: ' $warns | grep '%call_marker%' | cut -d

    while (<FUNCS>) {
        if ($_ =~ /(\d+) (.*)/) {
            if (int($1) > 200) {
                $too_common_funcs{$2} = 1;
            }
        }
    }

    close(FUNCS);

    open(FILE, ">", "$path/../$project.common_functions");
    foreach my $func (keys %too_common_funcs) {
        if ($func =~ / /) {
            next;
        }
        print FILE "$func\n";
    }
    close(FILE);
}

my $exec_name = $0;
my $path = $exec_name;
$path =~ s/(.*)\/.*/$1/;
my $project = shift;
my $warns = shift;
my $db_file = shift;

if (!defined($db_file)) {
    usage();
}

get_too_common_functions($path, $project, $warns);

my $db = DBI->connect("dbi:SQLite:$db_file", "", "", {AutoCommit => 0});
$db->do("PRAGMA cache_size = 800000");
$db->do("PRAGMA journal_mode = OFF");
$db->do("PRAGMA count_changes = OFF");
$db->do("PRAGMA temp_store = MEMORY");
$db->do("PRAGMA locking = EXCLUSIVE");
```



```
foreach my $func (keys %too_common_funcs) {
    $db->do("insert into common_caller_info values ('unknown', 'too common', '$f
}

my $call_id = 0;
my ($fn, $dummy, $sql);

open(WARNS, "<$warns");
while (<WARNS>) {
    # test.c:11 frob() SQL_caller_info: insert into caller_info values ('test.c'

    if (!($_ =~ /^.*? \w+\(\) SQL_caller_info: /)) {
        next;
    }
    ($dummy, $dummy, $dummy, $dummy, $dummy, $fn, $dummy) = split(/'/);

    if ($fn =~ /__builtin_/) {
        next;
    }
    if ($fn =~ /^(printk|memset|memcpy|kfree|printf|dev_err|writel)$/) {
        next;
    }

    ($dummy, $dummy, $sql) = split(/:/, $_, 3);

    if ($sql =~ /%call_marker%/) {
        $sql =~ s/%call_marker%//; # don't need this taking space in the db.
        $call_id++;
    }
    $sql =~ s/%CALL_ID%/$call_id/;

    $db->do($sql);
}
$db->commit();
$db->disconnect();
```

```
**********************************************************
     921 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/fill_db_sql.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use DBI;

my $project = shift;
my $warns = shift;
my $db_file = shift;

if (!defined($warns)) {
    print "usage:  $0 <-p=project> <smatch_warns.txt> <db_file>\n";
    exit(1);
}

my $db = DBI->connect("dbi:SQLite:$db_file", "", "", {AutoCommit => 0});
$db->do("PRAGMA cache_size = 800000");
$db->do("PRAGMA journal_mode = OFF");
$db->do("PRAGMA count_changes = OFF");
$db->do("PRAGMA temp_store = MEMORY");
$db->do("PRAGMA locking = EXCLUSIVE");

my ($dummy, $sql);

open(WARNS, "<$warns");
while (<WARNS>) {

    if (!($_ =~ /^.*? [^ ]*\(\) SQL: /)) {
        next;
    }
    ($dummy, $dummy, $sql) = split(/:/, $_, 3);

    $db->do($sql);
}
close(WARNS);

open(WARNS, "<$warns");
while (<WARNS>) {

    if (!($_ =~ /^.*? [^ ]*\(\) SQL_late: /)) {
        next;
    }
    ($dummy, $dummy, $sql) = split(/:/, $_, 3);

    $db->do($sql);
}
close(WARNS);

$db->commit();
$db->disconnect();
```

```
**********************************************************
    4092 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/fill_db_type_size.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use warnings;
use bigint;
use DBI;
use Data::Dumper;

my $project = shift;
my $warns = shift;
my $db_file = shift;
my $db = DBI->connect("dbi:SQLite:$db_file", "", "", {AutoCommit => 0});

my $raw_line;

sub text_to_int($)
{
    my $text = shift;

    if ($text =~ /s64min/) {
        return -(2**63);
    } elsif ($text =~/s32min/) {
        return -(2**31);
    } elsif ($text =~ /s16min/) {
        return -(2**15);
    } elsif ($text =~ /s64max/) {
        return 2**63 - 1;
    } elsif ($text =~ /s32max/) {
        return 2**31 - 1;
    } elsif ($text =~ /s16max/) {
        return 2**15 - 1;
    } elsif ($text =~ /u64max/) {
        return 2**62 - 1;
    } elsif ($text =~ /u32max/) {
        return 2**32 - 1;
    } elsif ($text =~ /u16max/) {
        return 2**16 - 1;
    }
    if ($text =~ /\((.*?)\)/) {
        $text = $1;
    }
    if (!($text =~ /^[-0123456789]/)) {
        return "NaN";
    }

    return int($text);
}

sub add_range($$$)
{
    my $union = shift;
    my $min = shift;
    my $max = shift;
    my %range;
    my @return_union;
    my $added = 0;
    my $check_next = 0;

    $range{min} = $min;
    $range{max} = $max;
```



```
    foreach my $tmp (@$union) {
        if ($added) {
            push @return_union, $tmp;
            next;
        }

        if ($range{max} < $tmp->{min}) {
            push @return_union, \%range;
            push @return_union, $tmp;
            $added = 1;
        } elsif ($range{min} <= $tmp->{min}) {
            if ($range{max} <= $tmp->{max}) {
                $range{max} = $tmp->{max};
                push @return_union, \%range;
                $added = 1;
            }
        } elsif ($range{min} <= $tmp->{max}) {
            if ($range{max} <= $tmp->{max}) {
                push @return_union, $tmp;
                $added = 1;
            } else {
                $range{min} = $tmp->{min};
            }
        } else {
            push @return_union, $tmp;
        }
    }

    if (!$added) {
        push @return_union, \%range;
    }

    return \@return_union;
}

sub print_num($)
{
    my $num = shift;

    if ($num < 0) {
        return "(" . $num . ")";
    } else {
        return $num;
    }
}

sub print_range($)
{
    my $range = shift;

    if ($range->{min} == $range->{max}) {
        return print_num($range->{min});
    } else {
        return print_num($range->{min}) . "-" .  print_num($range->{max});
    }
}

sub print_info($$)
{
    my $type = shift;
    my $union = shift;
    my $printed_range = "";
    my $i = 0;

    foreach my $range (@$union) {
```

```
127         if ($i) {
128             $printed_range = $printed_range . ",";
129         }
130         $i++;
131         $printed_range = $printed_range . print_range($range);
132     }
133     my $sql = "insert into type_size values ('$type', '$printed_range');";
134     $db->do($sql);
135 }


138 $db->do("PRAGMA cache_size = 800000");
139 $db->do("PRAGMA journal_mode = OFF");
140 $db->do("PRAGMA count_changes = OFF");
141 $db->do("PRAGMA temp_store = MEMORY");
142 $db->do("PRAGMA locking = EXCLUSIVE");

144 my ($sth, @row, $cur_type, $type, @ranges, $range_txt, %range, $min, $max, $unio

146 $sth = $db->prepare('select * from function_type_size order by type');
147 $sth->execute();

149 $skip = 0;
150 $cur_type = "";
151 while (@row = $sth->fetchrow_array()) {
152     $raw_line = join ',', @row;

154     $type = $row[2];

156     if ($cur_type ne "$type") {
157         if ($cur_type ne "" && $skip == 0) {
158             print_info($cur_type, $union_array);
159         }
160         $cur_type = $type;
161         $union_array = ();
162         $skip = 0;
163     }

165     @ranges = split(/,/, $row[3]);
166     foreach $range_txt (@ranges) {
167         if ($range_txt =~ /(.*[^(])-(.*)/) {
168             $min = text_to_int($1);
169             $max = text_to_int($2);
170         } else {
171             $min = text_to_int($range_txt);
172             $max = $min;
173         }
174         if ($min =~ /NaN/ || $max =~ /NaN/) {
175             $skip = 1;
176         }
177         $union_array = add_range($union_array, $min, $max);
178     }
179 }
180 if ($skip == 0) {
181     print_info($cur_type, $union_array);
182 }

184 $db->commit();
185 $db->disconnect();
```

```
**********************************************************
    4281 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/fill_db_type_value.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 #!/usr/bin/perl -w

  3 use strict;
  4 use warnings;
  5 use bigint;
  6 use DBI;
  7 use Data::Dumper;


 10 my $project = shift;
 11 my $warns = shift;
 12 my $db_file = shift;
 13 my $db = DBI->connect("dbi:SQLite:$db_file", "", "", {AutoCommit => 0});

 15 sub text_to_int($)
 16 {
 17     my $text = shift;

 19     if ($text =~ /s64min/) {
 20         return -(2**63);
 21     } elsif ($text =~/s32min/) {
 22         return -(2**31);
 23     } elsif ($text =~ /s16min/) {
 24         return -(2**15);
 25     } elsif ($text =~ /s64max/) {
 26         return 2**63 - 1;
 27     } elsif ($text =~ /s32max/) {
 28         return 2**31 - 1;
 29     } elsif ($text =~ /s16max/) {
 30         return 2**15 - 1;
 31     } elsif ($text =~ /u64max/) {
 32         return 2**64 - 1;
 33     } elsif ($text =~ /u32max/) {
 34         return 2**32 - 1;
 35     } elsif ($text =~ /u16max/) {
 36         return 2**16 - 1;
 37     }
 38     if ($text =~ /\((.*?)\)/) {
 39         $text = $1;
 40     }
 41     if (!($text =~ /^[-0123456789]/)) {
 42         return "NaN";
 43     }

 45     return int($text);
 46 }

 48 sub add_range($$$)
 49 {
 50     my $union = shift;
 51     my $min = shift;
 52     my $max = shift;
 53     my %range;
 54     my @return_union;
 55     my $added = 0;
 56     my $check_next = 0;

 58     $range{min} = $min;
 59     $range{max} = $max;
```



```
 61     foreach my $tmp (@$union) {
 62         if ($added) {
 63             push @return_union, $tmp;
 64             next;
 65         }

 67         if ($range{max} < $tmp->{min}) {
 68             push @return_union, \%range;
 69             push @return_union, $tmp;
 70             $added = 1;
 71         } elsif ($range{min} <= $tmp->{min}) {
 72             if ($range{max} <= $tmp->{max}) {
 73                 $range{max} = $tmp->{max};
 74                 push @return_union, \%range;
 75                 $added = 1;
 76             }
 77         } elsif ($range{min} <= $tmp->{max}) {
 78             if ($range{max} <= $tmp->{max}) {
 79                 push @return_union, $tmp;
 80                 $added = 1;
 81             } else {
 82                 $range{min} = $tmp->{min};
 83             }
 84         } else {
 85             push @return_union, $tmp;
 86         }
 87     }

 89     if (!$added) {
 90         push @return_union, \%range;
 91     }

 93     return \@return_union;
 94 }

 96 sub print_num($)
 97 {
 98     my $num = shift;

100     if ($num < 0) {
101         return "(" . $num . ")";
102     } else {
103         return $num;
104     }
105 }

107 sub print_range($)
108 {
109     my $range = shift;

111     if ($range->{min} == $range->{max}) {
112         return print_num($range->{min});
113     } else {
114         return print_num($range->{min}) . "-" .  print_num($range->{max});
115     }
116 }

118 sub print_info($$)
119 {
120     my $type = shift;
121     my $union = shift;
122     my $printed_range = "";
123     my $i = 0;

125     if ($#$union > 100) {
126         print "$type " . scalar @$union . "\n";
```

```
127         return;
128     }

130     foreach my $range (@$union) {
131         if ($i) {
132             $printed_range = $printed_range . ",";
133         }
134         $i++;
135         $printed_range = $printed_range . print_range($range);
136     }
137     my $sql = "insert into type_value values ('$type', '$printed_range');";
138     $db->do($sql);
139 }


142 $db->do("PRAGMA cache_size = 800000");
143 $db->do("PRAGMA journal_mode = OFF");
144 $db->do("PRAGMA count_changes = OFF");
145 $db->do("PRAGMA temp_store = MEMORY");
146 $db->do("PRAGMA locking = EXCLUSIVE");

148 my ($sth, @row, $cur_type, $type, @ranges, $range_txt, %range, $min, $max, $unio

150 $sth = $db->prepare('select type, value from function_type_value order by type')
151 $sth->execute();

153 $skip = 0;
154 $cur_type = "";
155 while (@row = $sth->fetchrow_array()) {
156     $type = $row[0];

158     if ($cur_type ne "$type") {
159         if ($cur_type ne "" && $skip == 0) {
160             print_info($cur_type, $union_array);
161         }
162         $cur_type = $type;
163         $union_array = ();
164         $skip = 0;
165     }

167     if ($skip == 1) {
168         next;
169     }

171     @ranges = split(/,/, $row[1]);
172     foreach $range_txt (@ranges) {
173         if ($range_txt =~ /ignore/) {
174             next;
175         }
176         if ($range_txt =~ /(.*[^(])-(.*)/) {
177             $min = text_to_int($1);
178             $max = text_to_int($2);
179         } else {
180             $min = text_to_int($range_txt);
181             $max = $min;
182         }
183         if ($min =~ /NaN/ || $max =~ /NaN/) {
184             $skip = 1;
185             last;
186         }
187         $union_array = add_range($union_array, $min, $max);
188     }
189 }
190 if ($skip == 0) {
191     print_info($cur_type, $union_array);
192 }
```



```
194 $db->commit();
195 $db->disconnect();
```

```
**********************************************************
     277 Fri Dec 21 15:00:17 2018
new/usr/src/tools/smatch/src/smatch_data/db/fixup_all.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

db_file=$1
cat << EOF | sqlite3 $db_file

delete from return_states where function = 'strlen';
delete from return_states where function = 'strnlen';
delete from return_states where function = 'sprintf';
delete from return_states where function = 'snprintf';

EOF
```

```
**********************************************************
   15306 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/fixup_kernel.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

db_file=$1
cat << EOF | sqlite3 $db_file
/* we only care about the main ->read/write() functions. */
delete from caller_info where function = '(struct file_operations)->read' and fi
delete from caller_info where function = '(struct file_operations)->write' and f
delete from caller_info where function = '(struct file_operations)->read' and ca
delete from caller_info where function = '(struct file_operations)->write' and c
delete from function_ptr where function = '(struct file_operations)->read';
delete from function_ptr where function = '(struct file_operations)->write';
delete from caller_info where function = '__vfs_write' and caller != 'vfs_write'
delete from caller_info where function = '__vfs_read' and caller != 'vfs_read';
delete from caller_info where function = '(struct file_operations)->write' and c
delete from caller_info where function = 'do_splice_from' and caller = 'direct_s

/* delete these function pointers which cause false positives */
delete from caller_info where function = '(struct file_operations)->open' and ty
delete from caller_info where function = '(struct notifier_block)->notifier_call
delete from caller_info where function = '(struct mISDNchannel)->send' and type
delete from caller_info where function = '(struct irq_router)->get' and type !=
delete from caller_info where function = '(struct irq_router)->set' and type !=
delete from caller_info where function = '(struct net_device_ops)->ndo_change_mt
delete from caller_info where function = '(struct timer_list)->function' and typ

/* type 1003 is USER_DATA */
delete from caller_info where caller = 'hid_input_report' and type = 1003;
delete from caller_info where caller = 'nes_process_iwarp_aeqe' and type = 1003;
delete from caller_info where caller = 'oz_process_ep0_urb' and type = 1003;
delete from caller_info where function = 'dev_hard_start_xmit' and key = '\$' an
delete from caller_info where function like '%->ndo_start_xmit' and key = '\$' a
delete from caller_info where caller = 'packet_rcv_fanout' and function = '(stru
delete from caller_info where caller = 'hptiop_probe' and type = 1003;
delete from caller_info where caller = 'p9_fd_poll' and function = '(struct file
delete from caller_info where caller = 'proc_reg_poll' and function = 'proc_reg_
delete from caller_info where function = 'blkdev_ioctl' and type = 1003 and para
/* 9017 is USER_DATA3_SET */
delete from return_states where function='vscnprintf' and type = 9017;
delete from return_states where function='scnprintf' and type = 9017;
delete from return_states where function='vsnprintf' and type = 9017;
delete from return_states where function='snprintf' and type = 9017;
delete from return_states where function='sprintf' and type = 9017;
delete from return_states where function='vscnprintf' and type = 8017;
delete from return_states where function='scnprintf' and type = 8017;
delete from return_states where function='vsnprintf' and type = 8017;
delete from return_states where function='snprintf' and type = 8017;
delete from return_states where function='sprintf' and type = 8017;
/* There is something setting skb->sk->sk_mark and friends to user_data and */
/* because of recursion it gets passed to everything and is impossible to debug
delete from caller_info where function = '__dev_queue_xmit' and type = 8017;
delete from caller_info where function = '__netdev_start_xmit' and type = 8017;
/* comparison doesn't deal with chunks, I guess.  */
delete from return_states where function='get_tty_driver' and type = 8017;
delete from caller_info where caller = 'snd_ctl_elem_write' and function = '(str
delete from caller_info where caller = 'snd_ctl_elem_read' and function = '(stru
delete from caller_info where function = 'nf_tables_newexpr' and type = 8017 and
delete from caller_info where caller = 'fb_set_var' and function = '(struct fb_o
delete from return_states where function = 'tty_lookup_driver' and parameter = 2

insert into caller_info values ('userspace', '', 'compat_sys_ioctl', 0, 0, 1003,
```



```
insert into caller_info values ('userspace', '', 'compat_sys_ioctl', 0, 0, 1003,
insert into caller_info values ('userspace', '', 'compat_sys_ioctl', 0, 0, 1003,

delete from caller_info where function = '(struct timer_list)->function' and par

/*
 * rw_verify_area is a very central function for the kernel.  The 1000000000
 * isn't accurate but I've picked it so that we can add "pos + count" without
 * wrapping on 32 bits.
 */
delete from return_states where function = 'rw_verify_area';
insert into return_states values ('faked', 'rw_verify_area', 0, 1, '0-1000000000
insert into return_states values ('faked', 'rw_verify_area', 0, 1, '0-1000000000
insert into return_states values ('faked', 'rw_verify_area', 0, 1, '0-1000000000
insert into return_states values ('faked', 'rw_verify_area', 0, 2, '(-4095)-(-1)

delete from return_states where function = 'is_kernel_rodata';
insert into return_states values ('faked', 'is_kernel_rodata', 0, 1, '1', 0, 0,
insert into return_states values ('faked', 'is_kernel_rodata', 0, 1, '1', 0, 103
insert into return_states values ('faked', 'is_kernel_rodata', 0, 2, '0', 0, 0,

/*
 * I am a bad person for doing this to __kmalloc() which is a very deep function
 * and can easily be removed instead of to kmalloc().  But kmalloc() is an
 * inline function so it ends up being recorded thousands of times in the
 * database.  Doing this is easier.
 *
 */
delete from return_states where function = '__kmalloc';
insert into return_states values ('faked', '__kmalloc', 0, 1, '16', 0,    0,  -1
insert into return_states values ('faked', '__kmalloc', 0, 1, '16', 0, 103,   0,
insert into return_states values ('faked', '__kmalloc', 0, 2, '0,500000000-57777
insert into return_states values ('faked', '__kmalloc', 0, 2, '0,500000000-57777
insert into return_states values ('faked', '__kmalloc', 0, 2, '0,500000000-57777
insert into return_states values ('faked', '__kmalloc', 0, 3, '0', 0,    0,  -1,
insert into return_states values ('faked', '__kmalloc', 0, 3, '0', 0,    103,  0

/*
 * Other kmalloc hacking.
 */
update return_states set return = '0,500000000-577777777' where function = 'kmal
update return_states set return = '0,500000000-577777777' where function = 'slab
update return_states set return = '0,500000000-577777777' where function = 'kmal
update return_states set return = '0,500000000-577777777' where function = 'kmal

delete from return_states where function = 'vmalloc';
insert into return_states values ('faked', 'vmalloc', 0, 1, '0,600000000-6777777
insert into return_states values ('faked', 'vmalloc', 0, 1, '0,600000000-6777777
insert into return_states values ('faked', 'vmalloc', 0, 2, '0', 0,    0,  -1, '

delete from return_states where function = 'ksize';
insert into return_states values ('faked', 'ksize', 0, 1, '0', 0,    0, -1, '',
insert into return_states values ('faked', 'ksize', 0, 1, '0', 0, 103,  0, '\$',
insert into return_states values ('faked', 'ksize', 0, 2, '1-4000000', 0,    0,

/* store a bunch of capped functions */
update return_states set return = '0-u32max[<=\$2]' where function = 'copy_to_us
update return_states set return = '0-u32max[<=\$2]' where function = '_copy_to_u
update return_states set return = '0-u32max[<=\$2]' where function = '__copy_to_
update return_states set return = '0-u32max[<=\$2]' where function = 'copy_from_
update return_states set return = '0-u32max[<=\$2]' where function = '_copy_from
update return_states set return = '0-u32max[<=\$2]' where function = '__copy_fro

update return_states set return = '0-8' where function = '__arch_hweight8';
update return_states set return = '0-16' where function = '__arch_hweight16';
update return_states set return = '0-32' where function = '__arch_hweight32';
```

```
update return_states set return = '0-64' where function = '__arch_hweight64';

/*
 * Preserve the value across byte swapping.  By the time we use it for math it
 * will be byte swapped back to CPU endian.
 */
update return_states set return = '0-u64max[==\$0]' where function = '__fswab64'
update return_states set return = '0-u32max[==\$0]' where function = '__fswab32'
update return_states set return = '0-u16max[==\$0]' where function = '__fswab16'
update return_states set return = '0-u64max[==\$0]' where function = '__builtin_
update return_states set return = '0-u32max[==\$0]' where function = '__builtin_
update return_states set return = '0-u16max[==\$0]' where function = '__builtin_

delete from return_states where function = 'bitmap_allocate_region' and return =
/* Just delete a lot of returns that everyone ignores */
delete from return_states where file = 'drivers/pci/access.c' and (return >= 129

update return_states set return = '(-4095)-s32max[<=\$1]' where function = 'get_
update return_states set return = '(-4095)-s64max[<=\$1]' where function = 'get_

/* Smatch can't parse wait_for_completion() */
update return_states set return = '(-108),(-22),0' where function = '__spi_sync'

delete from caller_info where caller = '__kernel_write';

/* We sometimes use pre-allocated 4097 byte buffers for performance critical cod
update caller_info set value = 4096 where caller='kernfs_file_direct_read' and f
/* let's pretend firewire doesn't exist */
delete from caller_info where caller='init_fw_attribute_group' and function='(st
/* and let's fake the next dev_attr_show() call entirely */
delete from caller_info where caller='sysfs_kf_seq_show' and function='(struct s
insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
insert into caller_info values ('fake', 'sysfs_kf_seq_show', '(struct sysfs_ops)
/* config fs confuses smatch a little */
update caller_info set value = 4096 where caller='fill_read_buffer' and function

/* smatch sees the memset() but not the subsequent changes */
update return_states set value = "" where function = 'gfs2_ea_find' and return =

delete from type_value where type = '(struct fd)->file';
delete from type_value where type = '(struct fd)->flags';

/* This is sometimes an enum or a u64 */
delete from type_value where type = '(struct mc_cmd_header)->status';

/* this is handled in check_kernel.c */
delete from return_states where function = "__write_once_size";

update return_states set value = "s32min-s32max[\$1]" where function = 'atomic_s

/* handled in the check itself */
delete from return_states where function = 'atomic_inc_return' and (type = 8023
delete from return_states where function = 'atomic_add_return' and (type = 8023
delete from return_states where function = 'atomic_sub_return' and (type = 8023
delete from return_states where function = 'atomic_sub_and_test' and (type = 802
delete from return_states where function = 'atomic_dec_and_test' and (type = 802
delete from return_states where function = 'atomic_dec' and (type = 8023 or type
delete from return_states where function = 'atomic_inc' and (type = 8023 or type
delete from return_states where function = 'atomic_sub' and (type = 8023 or type
delete from return_states where function = 'refcount_add_not_zero' and (type = 8
delete from return_states where function = 'refcount_inc_not_zero' and (type = 8
delete from return_states where function = 'refcount_sub_and_test' and (type = 8

update return_states set return = '0-32,2147483648-2147483690' where function =
```



```
update return_states set value = '0-u64max' where function = '_parse_integer' an

/* delete some function pointers which are sometimes byte units */
delete from caller_info where function = '(struct i2c_algorithm)->master_xfer' a

/* this if from READ_ONCE().  We can't know anything about the data.  */
delete from type_info where key = '(union anonymous)->__val';

EOF

# fixme: this is totally broken
call_id=$(echo "select distinct call_id from caller_info where function = '__ker
for id in $call_id ; do
    echo "insert into caller_info values ('fake', '', '__kernel_write', $id, 0,
done

for i in $(echo "select distinct return from return_states where function = 'cle
    echo "update return_states set return = \"$i[<=\$1]\" where return = \"$i\"
done

echo "select distinct file, function from function_ptr where ptr='(struct rtl_ha
        | sqlite3 $db_file | sed -e 's/|/ /' | while read file function ; do

    drv=$(echo $file | perl -ne 's/.*\/rtlwifi\/(.*?)\/sw.c/$1/; print')
    if [ $drv = "" ] ; then
        continue
    fi

    echo "update caller_info
          set function = '$drv (struct rtl_hal_ops)->set_hw_reg'
          where function = '(struct rtl_hal_ops)->set_hw_reg' and file like 'dri
         | sqlite3 $db_file

    echo "insert into function_ptr values ('$file', '$function', '$drv (struct r
         | sqlite3 $db_file
done
```

```
**********************************************************
       0 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/fixup_smatch_generic.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
     156 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/fn_data_link.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE fn_data_link (file varchar(128), function varchar(64), static boole
```

```
**********************************************************
     115 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/fn_ptr_data_link.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE fn_ptr_data_link (fn_ptr varchar(80), data varchar(80), CONSTRAINT
```

```
**********************************************************
     107 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/function_ptr.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE function_ptr (file varchar(128), function varchar(64), ptr varchar(
```

```
**********************************************************
     123 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/function_type.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE function_type (file varchar(80), function varchar(80), static boole
```

```
**********************************************************
     148 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/function_type_info.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE function_type_info (file varchar(128), function varchar(80), type i
```

```
**********************************************************
     111 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/function_type_size.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE function_type_size (file varchar(128), function varchar(80), type v
```

```
**********************************************************
     113 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/function_type_value.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE function_type_value (file varchar(128), function varchar(80), type
```

```
**********************************************************
    1670 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/init_constraints.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use warnings;
use bigint;
use DBI;
use Data::Dumper;
use File::Basename;
use Try::Tiny;

my $project = shift;
$project =~ s/.*=(.*)/$1/;
my $warns = shift;
my $db_file = shift;

sub preserve_existing_constraints()
{
    if (! -e "smatch_db.sqlite") {
        return;
    }

    my $db = DBI->connect("dbi:SQLite:$db_file", "", "",);
    $db->do('attach "smatch_db.sqlite" as old_db');
    $db->do('insert into constraints select * from old_db.constraints');
    $db->disconnect();
}

my $db;

sub connect_to_db($)
{
    my $name = shift;

    $db = DBI->connect("dbi:SQLite:$name", "", "", {AutoCommit => 0});

    $db->do("PRAGMA cache_size = 800000");
    $db->do("PRAGMA journal_mode = OFF");
    $db->do("PRAGMA count_changes = OFF");
    $db->do("PRAGMA temp_store = MEMORY");
    $db->do("PRAGMA locking = EXCLUSIVE");
}

sub load_manual_constraints($$)
{
    my $full_path = shift;
    my $project = shift;
    my $dir = dirname($full_path);

    open(FILE, "$dir/$project.constraints");
    while (<FILE>) {
        s/\n//;
        $db->do("insert or ignore into constraints (str) values ('$_')");
    }
    close(FILE);

    open(FILE, "$dir/$project.constraints_required");
    while (<FILE>) {
        my $limit;
        my $dummy;
```



```
        ($dummy, $dummy, $limit) = split(/,/);
        $limit =~ s/^ +//;
        $limit =~ s/\n//;
        try {
            $db->do("insert or ignore into constraints (str) values ('$limit')")
        } catch {}
    }
    close(FILE);

    $db->commit();
}

preserve_existing_constraints();

connect_to_db($db_file);
load_manual_constraints($0, $project);

$db->commit();
$db->disconnect();
```

```
**********************************************************
    1127 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/init_constraints_required.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use warnings;
use bigint;
use DBI;
use Data::Dumper;
use File::Basename;

my $project = shift;
$project =~ s/.*=(.*)/$1/;
my $warns = shift;
my $db_file = shift;

my $db;

sub connect_to_db($)
{
    my $name = shift;

    $db = DBI->connect("dbi:SQLite:$name", "", "", {AutoCommit => 0});

    $db->do("PRAGMA cache_size = 800000");
    $db->do("PRAGMA journal_mode = OFF");
    $db->do("PRAGMA count_changes = OFF");
    $db->do("PRAGMA temp_store = MEMORY");
    $db->do("PRAGMA locking = EXCLUSIVE");
}

sub load_manual_constraints($$)
{
    my $full_path = shift;
    my $project = shift;
    my $dir = dirname($full_path);
    my ($data, $op, $limit);

    open(FILE, "$dir/$project.constraints_required");
    while (<FILE>) {
        ($data, $op, $limit) = split(/,/);
        $op =~ s/ //g;
        $limit =~ s/^ +//;
        $limit =~ s/\n//;
        $db->do("insert into constraints_required values (?, ?, ?);", undef, $da
    }
    close(FILE);

    $db->commit();
}

connect_to_db($db_file);
load_manual_constraints($0, $project);

$db->commit();
$db->disconnect();
```

```
**********************************************************
       8 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/kernel.constraints
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 trusted
```

```
**********************************************************
     279 Fri Dec 21 15:00:18 2018
new/usr/src/tools/smatch/src/smatch_data/db/kernel.constraints_required
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 (struct seq_oss_devinfo)->synths, <, (struct seq_oss_devinfo)->max_synthdev
   2 (struct hid_report)->field, <, (struct hid_report)->maxfield
   3 (struct pidff_device)->pid_id, <, (struct ff_device)->max_effects
   4 (struct usb_mixer_elem_info)->cache_val, <, (struct snd_ctl_elem_id)->index
```

```
**********************************************************
    1634 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/kernel.return_fixes
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
i2c_master_recv s32min-s32max 1-s32max[<=$2]
i2c_master_recv s32min-0,2-s32max 1-s32max[<=$2]
hid_hw_output_report s32min-s32max 1-s32max[<=$2]
_regmap_read s32min-(-1),1-s32max (-4095)-(-1)
regmap_bulk_read s32min-(-1),1-s32max (-4095)-(-1)
scnprintf s32min-s32max 0-s32max[<$1]
scnprintf s32min-(-2),0-2147483646[<$1] 0-s32max[<$1]
scnprintf s32min-(-2),0-2147483646 0-s32max[<$1]
scnprintf s32min-s32max[<=$1] 0-s32max[<$1]
scnprintf 0-s32max 0-s32max[<$1]
vscnprintf s32min-(-2),0-s32max[<$1] 0-s32max[<$1]
down_interruptible s32min-s32max (-62),(-4)
__sock_create s32min-(-1),1-s32max (-4095)-(-1)
sock_create_kern s32min-(-1),1-s32max (-4095)-(-1)
nilfs_cpfile_get_checkpoint_block s32min-(-18),(-16)-s32max (-4095)-(-18),(-16)-
nilfs_cpfile_get_checkpoint_block s32min-(-18),(-16)-(-3),(-1),1-s32max (-4095)-
nilfs_mdt_insert_new_block s32min-(-23),(-21)-(-1),1-s32max (-4095)-(-23),(-21)-
simple_write_to_buffer s64min-s64max 0-s32max[<=$1]
atomic_read s32min-s32max s32min-s32max[==$0->counter]
notifier_to_errno (-2147483646)-(-1) (-4095)-(-1)
mc_status_to_error s32min-s32max (-4095)-0
dma_fence_wait_timeout s64min-s64max (-4095)-s64max
dma_fence_wait_timeout s32min-s32max (-4095)-s32max
fls s32min-s32max 0-32
fls64 s64min-s64max 0-64
__bitmap_weight s32min-s32max 0-s32max[<=$1]
__bitmap_weight 0-s32max 0-s32max[<=$1]
__ffs 0-u64max 0-63
__ffs 0-u32max 0-31
__spi_sync (-524),(-115),(-108),(-22) (-4095)-0
tpm_tis_spi_read_bytes s32min-s32max (-4095)-0
__irq_domain_activate_irq s32min-s32max (-4095)-0
get_user_pages_fast s32min-s32max 1-s32max[<$1]
__nci_request s32min-s32max (-4095)-0
```

```
**********************************************************
      89 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/local_values.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 CREATE TABLE local_values (file varchar(128), variable varchar(64), value varcha
```

```
**********************************************************
    1477 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/mark_function_ptrs_searchable.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use warnings;
use bigint;
use DBI;
use Data::Dumper;

my $db_file = shift;
my $db = DBI->connect("dbi:SQLite:$db_file", "", "", {AutoCommit => 0});

$db->do("PRAGMA cache_size = 800000");
$db->do("PRAGMA journal_mode = OFF");
$db->do("PRAGMA count_changes = OFF");
$db->do("PRAGMA temp_store = MEMORY");
$db->do("PRAGMA locking = EXCLUSIVE");

my ($update, $sth, $fn_ptr, $ptr_to_ptr, $count);

$update = $db->prepare_cached('UPDATE function_ptr set searchable = 1 where ptr
$sth = $db->prepare('select distinct(ptr) from function_ptr;');
$sth->execute();

while ($fn_ptr = $sth->fetchrow_array()) {

    # following a pointer to pointer chain is too complicated for now
    $ptr_to_ptr = $db->selectrow_array("select function from function_ptr where
    if ($ptr_to_ptr) {
        next;
    }
    $ptr_to_ptr = $db->selectrow_array("select function from function_ptr where
    if ($ptr_to_ptr) {
        next;
    }

    $count = $db->selectrow_array("select count(*) from return_states join funct
    # if there are too many states then bail
    if ($count > 1000) {
        next;
    }
    # if there are no states at all then don't bother recording
    if ($count == 0) {
        next;
    }

    $update->execute($fn_ptr);
}

$db->commit();
$db->disconnect();
```

```
**********************************************************
     140 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/mtag_about.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE mtag_about (tag big int, file varchar(80), function varchar(80), li
```

```
**********************************************************
      53 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/mtag_alias.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE mtag_alias (orig big int, new big int);
```

```
**********************************************************
      87 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/mtag_data.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE mtag_data (tag big int, offset integer, type integer, value varchar
```

```
**********************************************************
      72 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/mtag_map.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE mtag_map (tag big int, offset integer, container big int);
```

```
**********************************************************
     117 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/param_map.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE param_map (file varchar(256), to_from bool, fn_ptr varchar(80), par
```

```
**********************************************************
     124 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/parameter_name.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE parameter_name (file varchar(80), function varchar(80), static bool
```

```
**********************************************************
    1293 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/reload_partial.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

if echo $1 | grep -q '^-p' ; then
    PROJ=$(echo $1 | cut -d = -f 2)
    shift
fi

info_file=$1

if [[ "$info_file" = "" ]] ; then
    echo "Usage:  $0 -p=<project> <file with smatch messages>"
    exit 1
fi

bin_dir=$(dirname $0)
db_file=smatch_db.sqlite

files=$(grep "insert into caller_info" $info_file | cut -d : -f 1 | sort -u)
for c_file in $files; do
    echo "FILE $c_file"
    echo "delete from caller_info where file = '$c_file';" | sqlite3 $db_file
    echo "delete from return_states where file = '$c_file';" | sqlite3 $db_file
    echo "delete from call_implies where file = '$c_file';" | sqlite3 $db_file
    echo "delete from return_implies where file = '$c_file';" | sqlite3 $db_file
done

tmp_file=$(mktemp)

grep "insert into caller_info" $info_file > $tmp_file
${bin_dir}/fill_db_caller_info.pl "$PROJ" $tmp_file $db_file

grep "insert into return_states" $info_file > $tmp_file
${bin_dir}/fill_db_sql.pl "$PROJ" $tmp_file $db_file

grep "into call_implies" $info_file > $tmp_file
${bin_dir}/fill_db_sql.pl "$PROJ" $tmp_file $db_file

grep "into return_implies" $info_file > $tmp_file
${bin_dir}/fill_db_sql.pl "$PROJ" $tmp_file $db_file

rm $tmp_file

${bin_dir}/fixup_all.sh $db_file
if [ "$PROJ" != "" ] ; then
    ${bin_dir}/fixup_${PROJ}.sh $db_file
fi
```

```
**********************************************************
    1548 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/remove_mixed_up_pointer_params.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl -w

use strict;
use warnings;
use bigint;
use DBI;
use Data::Dumper;

my $db_file = shift;
my $db = DBI->connect("dbi:SQLite:$db_file", "", "", {AutoCommit => 0});

$db->do("PRAGMA cache_size = 800000");
$db->do("PRAGMA journal_mode = OFF");
$db->do("PRAGMA count_changes = OFF");
$db->do("PRAGMA temp_store = MEMORY");
$db->do("PRAGMA locking = EXCLUSIVE");

my ($select, $select_type, $remove, $file, $caller, $function, $param, $src_para

$remove = $db->prepare_cached('DELETE FROM caller_info WHERE file = ? AND caller
$select = $db->prepare('SELECT file, caller, function, parameter, value FROM cal
$select_type = $db->prepare_cached('SELECT value from function_type WHERE file =
$select->execute();

while (($file, $caller, $function, $param, $value) = $select->fetchrow_array())

    if ($value =~ /p (.*)/) {
        $src_param = $1;
    } else {
        print "error:  unexpected source parameter $value\n";
        next;
    }

    $select_type->execute($file, $caller, $src_param);
    $type = $select_type->fetchrow_array();
    if (!$type) {
        next;
    }
    #FIXME: Why is this extra fetch() needed???
    $select_type->fetch();

    if (!($type =~ /^void\*$/) && !($type =~ /^ulong$/)) {
        next;
    }

    $remove->execute($file, $caller, $function, $param);
}

$db->commit();
$db->disconnect();
```

```
**********************************************************
     281 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/return_implies.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
CREATE TABLE return_implies (
        file varchar(128),
        function varchar(64),
        call_id integer,
        static boolean,
        type integer,
        parameter integer,
        key varchar(256),
        value varchar(256),

        CONSTRAINT implies_row UNIQUE (file, function, call_id, static, type, pa
);
```

```
**********************************************************
     214 Fri Dec 21 15:00:19 2018
new/usr/src/tools/smatch/src/smatch_data/db/return_states.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE return_states (file varchar(128), function varchar(64), call_id int
```

```
**********************************************************
     135 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/db/sink_info.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE sink_info (file varchar(128), static boolean, sink_name varchar(64)
```

```
**********************************************************
   20742 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/db/smdb.py
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/python

# Copyright (C) 2013 Oracle.
#
# Licensed under the Open Software License version 1.1

import sqlite3
import sys
import re

try:
    con = sqlite3.connect('smatch_db.sqlite')
except sqlite3.Error, e:
    print "Error %s:" % e.args[0]
    sys.exit(1)

def usage():
    print "%s" %(sys.argv[0])
    print "<function> - how a function is called"
    print "return_states <function> - what a function returns"
    print "call_tree <function> - show the call tree"
    print "where <struct_type> <member> - where a struct member is set"
    print "type_size <struct_type> <member> - how a struct member is allocated"
    print "data_info <struct_type> <member> - information about a given data typ
    print "function_ptr <function> - which function pointers point to this"
    print "trace_param <function> <param> - trace where a parameter came from"
    print "locals <file> - print the local values in a file."
    sys.exit(1)

function_ptrs = []
searched_ptrs = []
def get_function_pointers_helper(func):
    cur = con.cursor()
    cur.execute("select distinct ptr from function_ptr where function = '%s';" %
    for row in cur:
        ptr = row[0]
        if ptr in function_ptrs:
            continue
        function_ptrs.append(ptr)
        if not ptr in searched_ptrs:
            searched_ptrs.append(ptr)
            get_function_pointers_helper(ptr)

def get_function_pointers(func):
    global function_ptrs
    global searched_ptrs
    function_ptrs = [func]
    searched_ptrs = [func]
    get_function_pointers_helper(func)
    return function_ptrs

db_types = {   0: "INTERNAL",
             101: "PARAM_CLEARED",
             103: "PARAM_LIMIT",
             104: "PARAM_FILTER",
            1001: "PARAM_VALUE",
            1002: "BUF_SIZE",
            1003: "USER_DATA",
            1004: "CAPPED_DATA",
            1005: "RETURN_VALUE",
```



```
            1006: "DEREFERENCE",
            1007: "RANGE_CAP",
            1008: "LOCK_HELD",
            1009: "LOCK_RELEASED",
            1010: "ABSOLUTE_LIMITS",
            1012: "PARAM_ADD",
            1013: "PARAM_FREED",
            1014: "DATA_SOURCE",
            1015: "FUZZY_MAX",
            1016: "STR_LEN",
            1017: "ARRAY_LEN",
            1018: "CAPABLE",
            1019: "NS_CAPABLE",
            1022: "TYPE_LINK",
            1023: "UNTRACKED_PARAM",
            1024: "CULL_PATH",
            1025: "PARAM_SET",
            1026: "PARAM_USED",
            1027: "BYTE_UNITS",
            1028: "COMPARE_LIMIT",
            1029: "PARAM_COMPARE",
            8017: "USER_DATA2",
            8018: "NO_OVERFLOW",
            8019: "NO_OVERFLOW_SIMPLE",
            8020: "LOCKED",
            8021: "UNLOCKED",
            8023: "ATOMIC_INC",
            8024: "ATOMIC_DEC",
};

def add_range(rl, min_val, max_val):
    check_next = 0
    done = 0
    ret = []
    idx = 0

    if len(rl) == 0:
        return [[min_val, max_val]]

    for idx in range(len(rl)):
        cur_min = rl[idx][0]
        cur_max = rl[idx][1]

        # we already merged the new range but we might need to change later
        # ranges if they over lap with more than one
        if check_next:
            # join with added range
            if max_val + 1 == cur_min:
                ret[len(ret) - 1][1] = cur_max
                done = 1
                break
            # don't overlap
            if max_val < cur_min:
                ret.append([cur_min, cur_max])
                done = 1
                break
            # partially overlap
            if max_val < cur_max:
                ret[len(ret) - 1][1] = cur_max
                done = 1
                break
            # completely overlap
            continue

        # join 2 ranges into one
        if max_val + 1 == cur_min:
```

```
127             ret.append([min_val, cur_max])
128             done = 1
129             break
130         # range is entirely below
131         if max_val < cur_min:
132             ret.append([min_val, max_val])
133             ret.append([cur_min, cur_max])
134             done = 1
135             break
136         # range is partially below
137         if min_val < cur_min:
138             if max_val <= cur_max:
139                 ret.append([min_val, cur_max])
140                 done = 1
141                 break
142             else:
143                 ret.append([min_val, max_val])
144                 check_next = 1
145                 continue
146         # range already included
147         if max_val <= cur_max:
148             ret.append([cur_min, cur_max])
149             done = 1
150             break;
151         # range partially above
152         if min_val <= cur_max:
153             ret.append([cur_min, max_val])
154             check_next = 1
155             continue
156         # join 2 ranges on the other side
157         if min_val - 1 == cur_max:
158             ret.append([cur_min, max_val])
159             check_next = 1
160             continue
161         # range is above
162         ret.append([cur_min, cur_max])

164     if idx + 1 < len(rl):           # we hit a break statement
165         ret = ret + rl[idx + 1:]
166     elif done:                      # we hit a break on the last iteration
167         pass
168     elif not check_next:            # it's past the end of the rl
169         ret.append([min_val, max_val])

171     return ret;

173 def rl_union(rl1, rl2):
174     ret = []
175     for r in rl1:
176         ret = add_range(ret, r[0], r[1])
177     for r in rl2:
178         ret = add_range(ret, r[0], r[1])

180     if (rl1 or rl2) and not ret:
181         print "bug: merging %s + %s gives empty" %(rl1, rl2)

183     return ret

185 def txt_to_val(txt):
186     if txt == "s64min":
187         return -(2**63)
188     elif txt == "s32min":
189         return -(2**31)
190     elif txt == "s16min":
191         return -(2**15)
192     elif txt == "s64max":
```



```
193         return 2**63 - 1
194     elif txt == "s32max":
195         return 2**31 - 1
196     elif txt == "s16max":
197         return 2**15 - 1
198     elif txt == "u64max":
199         return 2**64 - 1
200     elif txt == "u32max":
201         return 2**32 - 1
202     elif txt == "u16max":
203         return 2**16 - 1
204     else:
205         try:
206             return int(txt)
207         except ValueError:
208             return 0

210 def val_to_txt(val):
211     if val == -(2**63):
212         return "s64min"
213     elif val == -(2**31):
214         return "s32min"
215     elif val == -(2**15):
216         return "s16min"
217     elif val == 2**63 - 1:
218         return "s64max"
219     elif val == 2**31 - 1:
220         return "s32max"
221     elif val == 2**15 - 1:
222         return "s16max"
223     elif val == 2**64 - 1:
224         return "u64max"
225     elif val == 2**32 - 1:
226         return "u32max"
227     elif val == 2**16 - 1:
228         return "u16max"
229     elif val < 0:
230         return "(%d)" %(val)
231     else:
232         return "%d" %(val)

234 def get_next_str(txt):
235     val = ""
236     parsed = 0

238     if txt[0] == '(':
239         parsed += 1
240         for char in txt[1:]:
241             if char == ')':
242                 break
243             parsed += 1
244         val = txt[1:parsed]
245         parsed += 1
246     elif txt[0] == 's' or txt[0] == 'u':
247         parsed += 6
248         val = txt[:parsed]
249     else:
250         if txt[0] == '-':
251             parsed += 1
252         for char in txt[parsed:]:
253             if char == '-':
254                 break
255             parsed += 1
256         val = txt[:parsed]
257     return [parsed, val]
```

```
259 def txt_to_rl(txt):
260     if len(txt) == 0:
261         return []

263     ret = []
264     pairs = txt.split(",")
265     for pair in pairs:
266         cnt, min_str = get_next_str(pair)
267         if cnt == len(pair):
268             max_str = min_str
269         else:
270             cnt, max_str = get_next_str(pair[cnt + 1:])
271         min_val = txt_to_val(min_str)
272         max_val = txt_to_val(max_str)
273         ret.append([min_val, max_val])

275 #    Hm...  Smatch won't call INT_MAX s32max if the variable is unsigned.
276 #    if txt != rl_to_txt(ret):
277 #        print "bug: converting: text = %s rl = %s internal = %s" %(txt, rl_to_t

279     return ret

281 def rl_to_txt(rl):
282     ret = ""
283     for idx in range(len(rl)):
284         cur_min = rl[idx][0]
285         cur_max = rl[idx][1]

287         if idx != 0:
288             ret += ","

290         if cur_min == cur_max:
291             ret += val_to_txt(cur_min)
292         else:
293             ret += val_to_txt(cur_min)
294             ret += "-"
295             ret += val_to_txt(cur_max)
296     return ret

298 def type_to_str(type_int):

300     t = int(type_int)
301     if db_types.has_key(t):
302         return db_types[t]
303     return type_int

305 def type_to_int(type_string):
306     for k in db_types.keys():
307         if db_types[k] == type_string:
308             return k
309     return -1

311 def display_caller_info(printed, cur, param_names):
312     for txt in cur:
313         if not printed:
314             print "file | caller | function | type | parameter | key | value |"
315         printed = 1

317         parameter = int(txt[6])
318         key = txt[7]
319         if len(param_names) and parameter in param_names:
320             key = key.replace("$", param_names[parameter])

322         print "%20s | %20s | %20s |" %(txt[0], txt[1], txt[2]),
323         print " %10s |" %(type_to_str(txt[5])),
324         print " %d | %s | %s" %(parameter, key, txt[8])
```



```
325     return printed

327 def get_caller_info(filename, ptrs, my_type):
328     cur = con.cursor()
329     param_names = get_param_names(filename, func)
330     printed = 0
331     type_filter = ""
332     if my_type != "":
333         type_filter = "and type = %d" %(type_to_int(my_type))
334     for ptr in ptrs:
335         cur.execute("select * from caller_info where function = '%s' %s;" %(ptr,
336         printed = display_caller_info(printed, cur, param_names)

338 def print_caller_info(filename, func, my_type = ""):
339     ptrs = get_function_pointers(func)
340     get_caller_info(filename, ptrs, my_type)

342 def merge_values(param_names, vals, cur):
343     for txt in cur:
344         parameter = int(txt[0])
345         name = txt[1]
346         rl = txt_to_rl(txt[2])
347         if parameter in param_names:
348             name = name.replace("$", param_names[parameter])

350         if not parameter in vals:
351             vals[parameter] = {}

353         # the first item on the list is the number of rows.  it's incremented
354         # every time we call merge_values().
355         if name in vals[parameter]:
356             vals[parameter][name] = [vals[parameter][name][0] + 1, rl_union(vals
357         else:
358             vals[parameter][name] = [1, rl]

360 def get_param_names(filename, func):
361     cur = con.cursor()
362     param_names = {}
363     cur.execute("select parameter, value from parameter_name where file = '%s' a
364     for txt in cur:
365         parameter = int(txt[0])
366         name = txt[1]
367         param_names[parameter] = name
368     if len(param_names):
369         return param_names

371     cur.execute("select parameter, value from parameter_name where function = '%
372     for txt in cur:
373         parameter = int(txt[0])
374         name = txt[1]
375         param_names[parameter] = name
376     return param_names

378 def get_caller_count(ptrs):
379     cur = con.cursor()
380     count = 0
381     for ptr in ptrs:
382         cur.execute("select count(distinct(call_id)) from caller_info where func
383         for txt in cur:
384             count += int(txt[0])
385     return count

387 def print_merged_caller_values(filename, func, ptrs, param_names, call_cnt):
388     cur = con.cursor()
389     vals = {}
390     for ptr in ptrs:
```

```
391         cur.execute("select parameter, key, value from caller_info where functio
392         merge_values(param_names, vals, cur);

394     for param in sorted(vals):
395         for name in sorted(vals[param]):
396             if vals[param][name][0] != call_cnt:
397                 continue
398             print "%d %s -> %s" %(param, name, rl_to_txt(vals[param][name][1]))


401 def print_unmerged_caller_values(filename, func, ptrs, param_names):
402     cur = con.cursor()
403     for ptr in ptrs:
404         prev = -1
405         cur.execute("select file, caller, call_id, parameter, key, value from ca
406         for filename, caller, call_id, parameter, name, value in cur:
407             if prev != int(call_id):
408                 prev = int(call_id)

410             parameter = int(parameter)
411             if parameter < len(param_names):
412                 name = name.replace("$", param_names[parameter])
413             else:
414                 name = name.replace("$", "$%d" %(parameter))

416             print "%s | %s | %s | %s" %(filename, caller, name, value)
417         print "========================="

419 def print_caller_values(filename, func, ptrs):
420     param_names = get_param_names(filename, func)
421     call_cnt = get_caller_count(ptrs)

423     print_merged_caller_values(filename, func, ptrs, param_names, call_cnt)
424     print "========================="
425     print_unmerged_caller_values(filename, func, ptrs, param_names)

427 def caller_info_values(filename, func):
428     ptrs = get_function_pointers(func)
429     print_caller_values(filename, func, ptrs)

431 def print_return_states(func):
432     cur = con.cursor()
433     cur.execute("select * from return_states where function = '%s';" %(func))
434     count = 0
435     for txt in cur:
436         printed = 1
437         if count == 0:
438             print "file | function | return_id | return_value | type | param | k
439         count += 1
440         print "%s | %s | %2s | %13s" %(txt[0], txt[1], txt[3], txt[4]),
441         print "| %13s |" %(type_to_str(txt[6])),
442         print " %2d | %20s | %20s |" %(txt[7], txt[8], txt[9])

444 def print_return_implies(func):
445     cur = con.cursor()
446     cur.execute("select * from return_implies where function = '%s';" %(func))
447     count = 0
448     for txt in cur:
449         if not count:
450             print "file | function | type | param | key | value |"
451         count += 1
452         print "%15s | %15s" %(txt[0], txt[1]),
453         print "| %15s" %(type_to_str(txt[4])),
454         print "| %3d | %s | %15s |" %(txt[5], txt[6], txt[7])

456 def print_type_size(struct_type, member):
```



```
457     cur = con.cursor()
458     cur.execute("select * from type_size where type like '(struct %s)->%s';" %(s
459     print "type | size"
460     for txt in cur:
461         print "%-15s | %s" %(txt[0], txt[1])

463     cur.execute("select * from function_type_size where type like '(struct %s)->
464     print "file | function | type | size"
465     for txt in cur:
466         print "%-15s | %-15s | %-15s | %s" %(txt[0], txt[1], txt[2], txt[3])

468 def print_data_info(struct_type, member):
469     cur = con.cursor()
470     cur.execute("select * from data_info where data like '(struct %s)->%s';" %(s
471     print "file | data | type | value"
472     for txt in cur:
473         print "%-15s | %-15s | %-15s | %s" %(txt[0], txt[1], type_to_str(txt[2])

475 def print_fn_ptrs(func):
476     ptrs = get_function_pointers(func)
477     if not ptrs:
478         return
479     print "%s = " %(func),
480     print(ptrs)

482 def print_functions(member):
483     cur = con.cursor()
484     cur.execute("select * from function_ptr where ptr like '%%->%s';" %(member))
485     print "File | Pointer | Function | Static"
486     for txt in cur:
487         print "%-15s | %-15s | %-15s | %s" %(txt[0], txt[2], txt[1], txt[3])

489 def get_callers(func):
490     ret = []
491     cur = con.cursor()
492     ptrs = get_function_pointers(func)
493     for ptr in ptrs:
494         cur.execute("select distinct caller from caller_info where function = '%
495         for row in cur:
496             ret.append(row[0])
497     return ret

499 printed_funcs = []
500 def call_tree_helper(func, indent = 0):
501     global printed_funcs
502     if func in printed_funcs:
503         return
504     print "%s%s()" %(" " * indent, func)
505     if func == "too common":
506         return
507     if indent > 6:
508         return
509     printed_funcs.append(func)
510     callers = get_callers(func)
511     if len(callers) >= 20:
512         print "Over 20 callers for %s()" %(func)
513         return
514     for caller in callers:
515         call_tree_helper(caller, indent + 2)

517 def print_call_tree(func):
518     global printed_funcs
519     printed_funcs = []
520     call_tree_helper(func)

522 def function_type_value(struct_type, member):
```

```
523     cur = con.cursor()
524     cur.execute("select * from function_type_value where type like '(struct %s)-
525     for txt in cur:
526         print "%-30s | %-30s | %s | %s" %(txt[0], txt[1], txt[2], txt[3])

528 def trace_callers(func, param):
529     sources = []
530     prev_type = 0

532     cur = con.cursor()
533     ptrs = get_function_pointers(func)
534     for ptr in ptrs:
535         cur.execute("select type, caller, value from caller_info where function
536         for row in cur:
537             data_type = int(row[0])
538             if data_type == 1014:
539                 sources.append((row[1], row[2]))
540             elif data_type == 1028:
541                 sources.append(("%", row[2])) # hack...
542             elif data_type == 0 and prev_type == 0:
543                 sources.append((row[1], ""))
544             prev_type = data_type
545     return sources

547 def trace_param_helper(func, param, indent = 0):
548     global printed_funcs
549     if func in printed_funcs:
550         return
551     print "%s%s(param %d)" %(" " * indent, func, param)
552     if func == "too common":
553         return
554     if indent > 20:
555         return
556     printed_funcs.append(func)
557     sources = trace_callers(func, param)
558     for path in sources:

560         if len(path[1]) and path[1][0] == 'p' and path[1][1] == ' ':
561             p = int(path[1][2:])
562             trace_param_helper(path[0], p, indent + 2)
563         elif len(path[0]) and path[0][0] == '%':
564             print "  %s%s" %(" " * indent, path[1])
565         else:
566             print "* %s%s %s" %(" " * (indent - 1), path[0], path[1])

568 def trace_param(func, param):
569     global printed_funcs
570     printed_funcs = []
571     print "tracing %s %d" %(func, param)
572     trace_param_helper(func, param)

574 def print_locals(filename):
575     cur = con.cursor()
576     cur.execute("select file,data,value from data_info where file = '%s' and typ
577     for txt in cur:
578         print "%s | %s | %s" %(txt[0], txt[1], txt[2])

580 def constraint(struct_type, member):
581     cur = con.cursor()
582     cur.execute("select * from constraints_required where data like '(struct %s)
583     for txt in cur:
584         print "%-30s | %-30s | %s | %s" %(txt[0], txt[1], txt[2], txt[3])

586 if len(sys.argv) < 2:
587     usage()
```



```
589 if len(sys.argv) == 2:
590     func = sys.argv[1]
591     print_caller_info("", func)
592 elif sys.argv[1] == "call_info":
593     if len(sys.argv) != 4:
594         usage()
595     filename = sys.argv[2]
596     func = sys.argv[3]
597     caller_info_values(filename, func)
598     print_caller_info(filename, func)
599 elif sys.argv[1] == "user_data":
600     func = sys.argv[2]
601     print_caller_info(filename, func, "USER_DATA")
602 elif sys.argv[1] == "param_value":
603     func = sys.argv[2]
604     print_caller_info(filename, func, "PARAM_VALUE")
605 elif sys.argv[1] == "function_ptr" or sys.argv[1] == "fn_ptr":
606     func = sys.argv[2]
607     print_fn_ptrs(func)
608 elif sys.argv[1] == "return_states":
609     func = sys.argv[2]
610     print_return_states(func)
611     print "================================================"
612     print_return_implies(func)
613 elif sys.argv[1] == "return_implies":
614     func = sys.argv[2]
615     print_return_implies(func)
616 elif sys.argv[1] == "type_size" or sys.argv[1] == "buf_size":
617     struct_type = sys.argv[2]
618     member = sys.argv[3]
619     print_type_size(struct_type, member)
620 elif sys.argv[1] == "data_info":
621     struct_type = sys.argv[2]
622     member = sys.argv[3]
623     print_data_info(struct_type, member)
624 elif sys.argv[1] == "call_tree":
625     func = sys.argv[2]
626     print_call_tree(func)
627 elif sys.argv[1] == "where":
628     if len(sys.argv) == 3:
629         struct_type = "%"
630         member = sys.argv[2]
631     elif len(sys.argv) == 4:
632         struct_type = sys.argv[2]
633         member = sys.argv[3]
634     function_type_value(struct_type, member)
635 elif sys.argv[1] == "local":
636     filename = sys.argv[2]
637     variable = ""
638     if len(sys.argv) == 4:
639         variable = sys.argv[3]
640     local_values(filename, variable)
641 elif sys.argv[1] == "functions":
642     member = sys.argv[2]
643     print_functions(member)
644 elif sys.argv[1] == "trace_param":
645     if len(sys.argv) != 4:
646         usage()
647     func = sys.argv[2]
648     param = int(sys.argv[3])
649     trace_param(func, param)
650 elif sys.argv[1] == "locals":
651     if len(sys.argv) != 3:
652         usage()
653     filename = sys.argv[2]
654     print_locals(filename);
```

```
655 elif sys.argv[1] == "constraint":
656     if len(sys.argv) == 3:
657         struct_type = "%"
658         member = sys.argv[2]
659     elif len(sys.argv) == 4:
660         struct_type = sys.argv[2]
661         member = sys.argv[3]
662     constraint(struct_type, member)
663 elif sys.argv[1] == "test":
664     filename = sys.argv[2]
665     func = sys.argv[3]
666     caller_info_values(filename, func)
667 else:
668     usage()
```

```
**********************************************************
     161 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/db/type_info.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
CREATE TABLE type_info (
        file varchar(80),
        type integer,
        key varchar(80),
        value varchar(80),

        CONSTRAINT type_info_constraint UNIQUE (type, key, value)
);
```

```
**********************************************************
      57 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/db/type_size.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE type_size (type varchar(80), size integer);
```

```
**********************************************************
      63 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/db/type_value.schema
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 CREATE TABLE type_value (type varchar(80), value varchar(80));
```

```
**********************************************************
     662 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/db/vim_smdb
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

# Add these lines to your .vimrc file
#
# map <C-r> :! vim_smdb return_states <cword> <CR> :execute 'edit' system("cat ~
# map <C-c> :! vim_smdb <cword> <CR> :execute 'edit' system("cat ~/.smdb_tmp/cur
#
# Now you can move your cursor over a function and hit CTRL-c to see how it's
# called or CTRL-r to see what it returns.  Use the ":bd" command to get back to
# your source.

DIR="$HOME/.smdb_tmp"
mkdir -p $DIR

for i in $(seq 1 100) ; do
        if [ ! -e $DIR/$i ] ; then
                break
        fi
done

if [ $i == 100 ] ; then
        i=1
fi

next=$(($i + 1))

rm -f $DIR/$next
rm $DIR/.${i}.swp
smdb $* > $DIR/$i

echo "$DIR/$i" > $DIR/cur
```

**********
*332 Fri Dec 21 15:00:20 2018*
*new/usr/src/tools/smatch/src/smatch_data/illumos_kernel.no_return_funcs*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * These functions don't return.  Unfortunately, sparse today doesn't correctly
 * respect __NORETURN, so we have to explicitly list them here for now.
 */
__assert_fail
__builtin_unreachable
bop_panic
efi_reset
panic
pc_reset
prom_exit_to_mon
prom_panic
prom_reboot
reset
resume_from_zombie
swtch_from_zombie
thread_exit
vpanic
```

```
**********************************************************
     134 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/illumos_kernel.skipped_functions
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
/* These are "too hairy" for smatch. */
dtrace_disx86
elf32exec
elfexec
iscsi_ioctl
lm_idle_chk
segvn_fault_vnodepages
tcp_input_data
```

**********
332 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/illumos_user.no_return_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
/*
 * These functions don't return.  Unfortunately, sparse today doesn't correctly
 * respect __NORETURN, so we have to explicitly list them here for now.
 */
__assert_fail
__builtin_unreachable
abort
ctfmerge_fatal
err
errx
_exit
exit
_longjmp
longjmp
pthread_exit
reparsed_door_call_error
siglongjmp
terminate
thr_exit
verr
verrx
```

```
**********
     640 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/illumos_user.skipped_functions
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********
```

```
/*
 * The below functions cause smatch to fail with "turning off implications after
 * 60 seconds" or similar, generally because they're too large for it to handle.
 *
 * This will disable analysis altogether.
 */

/* libast */
_ast_optget
_ast_opthelp
/* libcmd */
b_uname
/* libcurses */
_updateln
/* libdisasm */
dtrace_disx86
/* libld */
ld32_sym_process
ld64_sym_process
update_osym
/* libsqlite */
sqliteVdbeExec
/* cmd/acpi/iasl */
AslCompilerparse
/* cmd/fs.d/autofs */
nfsmount
/* cmd/pppd */
lcp_nakci

/* generated code */
ipf_yyparse
ipmon_yyparse
ipnat_yyparse
ippool_yyparse
ndr__ndr_hdr
yyerror
yylex
yylook
yyparse
yywinput
```

```
**********************************************************
   23756 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.allocation_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of functions that return a new allocation.
// generated by 'gen_allocation_list.sh'
__a2mp_build
aa_alloc_profile
__aa_kvmalloc
aarp_alloc
ablkcipher_request_alloc
acpi_alloc_ipmi_msg
acpi_processor_alloc_pdc
acquire_group
add_device
add_missing_dev
add_vol
adfs_read_map
aer_alloc_rpc
afs_alloc_flat_call
afs_alloc_server
afs_vlocation_alloc
aggr_init
agp_alloc_bridge
agp_create_client
agp_create_controller
agp_create_memory
agp_create_user_memory
ahc_alloc
ahc_alloc_tstate
ahd_alloc
ahd_alloc_tstate
ai_attach
alloc_ai
alloc_apertures
alloc_area
alloc_async
allocate_bitmap_node
__allocate_fw_buf
allocate_partitions
allocate_request
__alloc_atm_dev
alloc_backref_edge
alloc_backref_node
alloc_buf
alloc_buffer
alloc_cell
alloc_channel
alloc_chunk
alloc_cld_upcall
alloc_client
alloc_conn
alloc_context
alloc_cpu_rmap
alloc_ctrl_packet
alloc_data_packet
alloc_dca_provider
alloc_dev
alloc_dev_data
alloc_ep
alloc_fdtable
__alloc_filter
alloc_flex_gd
alloc_ftrace_hash
```



```
alloc_fw_cache_entry
alloc_group_attrs
alloc_i7core_dev
alloc_jh
alloc_journal_list
alloc_lvn
alloc_mpc
alloc_mr
alloc_msg
alloc_msi_entry
alloc_multipath
alloc_namespace
alloc_net_device
alloc_nilfs
alloc_parallel
_alloc_path_selector
alloc_pcie_link_state
alloc_pci_root_info
alloc_pdev
alloc_perf_context
alloc_pgpath
alloc_pg_vec
alloc_pipe_info
alloc_pl
alloc_priority_group
alloc_pvd
alloc_rdma
alloc_read_gpt_entries
alloc_read_gpt_header
alloc_reloc_control
alloc_resource
alloc_rootdomain
alloc_rx_struct
alloc_sbridge_dev
alloc_sched_domains
alloc_scq
alloc_selector
alloc_session
alloc_smp_req
alloc_sock_iocb
alloc_super
alloc_switch_ctx
alloc_symbol_cache
alloc_tree
alloc_tx_struct
alloc_ubifs_info
alloc_uprobe
alloc_urbs
alloc_watch_adapter
alloc_worker
alloc_workqueue_attrs
__alloc_workqueue_key
amp_ctrl_add
amp_mgr_create
aoedev_by_aoeaddr
ap_add_sta
ap_auth_make_challenge
append_internal
array_zalloc
ast_i2c_create
ast_ttm_tt_create
ata_port_alloc
ath10k_ce_init_state
ath10k_htt_attach
ath6kl_htc_mbox_create
ath6kl_htc_pipe_create
```

```
ath6kl_usb_create
ath6kl_wmi_init
ath9k_htc_hw_alloc
ath9k_init_wmi
ath_gen_timer_alloc
atif_add_device
atom_parse
audit_alloc_context
audit_alloc_name
audit_buffer_alloc
audit_init_entry
audit_krule_to_data
autofs4_new_ino
avmcard_dma_alloc
ax25_create_cb
b1_alloc_card
b43_bus_dev_bcma_init
b43_bus_dev_ssb_init
b43_calibrate_lo_setting
b43_generate_dyn_tssi2dbm_tab
b43legacy_generate_probe_resp
b43legacy_setup_dmaring
b43legacy_setup_pioqueue
b43_setup_dmaring
b43_setup_pioqueue_rx
b43_setup_pioqueue_tx
batadv_dat_select_candidates
batadv_hardif_add_interface
batadv_hash_new
batadv_neigh_node_new
batadv_new_tt_req_node
bch_alloc
bch_cache_set_alloc
bfad_fcxp_map_sg
binder_get_thread
binder_new_node
bin_to_hex_dup
bio_alloc_map_data
bio_integrity_alloc
bioset_create
bl_alloc_extent
__blk_queue_init_tags
bnx2fc_alloc_work
bnx2fc_cmd_mgr_alloc
bnx2fc_hba_create
bnx2x_vfop_add
brcmf_sdbrcm_probe
brcmf_usbdev_qinit
brcms_c_ampdu_attach
brcms_c_antsel_attach
brcms_c_attach_malloc
brcms_c_bsscfg_malloc
brcms_c_channel_mgr_attach
brcms_init_timer
brd_alloc
br_multicast_new_port_group
bsd_alloc
bsg_alloc_device
btmrvl_add_card
btrfs_alloc_block_rsv
btrfs_alloc_delayed_item
btrfs_alloc_root
btrfsic_block_alloc
btrfsic_block_link_alloc
btrfsic_dev_state_alloc
btrfsic_stack_frame_alloc
```



```
build_htc_txctrl_packet
build_path_from_dentry
c4iw_uld_add
cachefiles_cook_key
caif_device_alloc
call_usermodehelper_setup
capiminor_alloc
capincci_alloc
carl9170_cmd_buf
cas_page_alloc
cciss_allocate_sg_chain_blocks
cdebbuf_alloc
cdev_alloc
cdv_intel_crtc_mode_get
ceph_buffer_new
ceph_create_snap_context
cfcnfg_create
cffrml_create
cfi_read_pri
cfi_staa_setup
cfusbl_create
cfv_alloc_and_copy_to_shm
cgroup_alloc_name
channel_detector_create
cifs_build_path_to_root
cifs_lock_init
cifs_new_fileinfo
cifs_readdata_alloc
cifs_strndup_from_utf16
cifs_strndup_to_utf16
cifs_writedata_alloc
cirrus_ttm_tt_create
class_compat_register
clusterip_config_init
cmtp_application_add
cnic_alloc_dev
cn_queue_alloc_callback_entry
cn_queue_alloc_dev
composite_default_mfr
conn_create
convert_chmap
copy_tlv
core_dev_init_initiator_node_lun_acl
cpia2_init_camera_struct
create_a
create_bounce_buffer
create_driver
create_durable_buf
create_event
create_l2
create_lease_buf
create_new_subsystem
create_osd
create_port
create_portdata0
create_reconnect_durable_buf
create_regulator
create_rmpp_recv
create_serial
create_trace_option_files
create_unique_id
create_uts_ns
create_wlan
__crypto_alloc_tfm
crypto_create_tfm
cs46xx_dsp_spos_create
```

```
cs5535_mfgpt_alloc_timer
cs_alloc_spec
csio_hw_alloc
CsrWifiEventCsrUint16CsrUint8Des
CsrWifiEventCsrUint16Des
CsrWifiEventCsrUint32Des
CsrWifiEventCsrUint8Des
CsrWifiEventDes
CsrWifiNmeApConfigSetReqDes
CsrWifiNmeApStaRemoveReqDes
CsrWifiNmeApStartCfmDes
CsrWifiNmeApStartReqDes
CsrWifiNmeApStationIndDes
CsrWifiNmeApStopCfmDes
CsrWifiNmeApStopIndDes
CsrWifiNmeApWmmParamUpdateReqDes
CsrWifiNmeApWpsRegisterCfmDes
CsrWifiNmeApWpsRegisterReqDes
CsrWifiRouterCtrlBlockAckDisableCfmDes
CsrWifiRouterCtrlBlockAckDisableReqDes
CsrWifiRouterCtrlBlockAckEnableCfmDes
CsrWifiRouterCtrlBlockAckEnableReqDes
CsrWifiRouterCtrlBlockAckErrorIndDes
CsrWifiRouterCtrlCapabilitiesCfmDes
CsrWifiRouterCtrlConfigurePowerModeReqDes
CsrWifiRouterCtrlConnectedIndDes
CsrWifiRouterCtrlHipIndDes
CsrWifiRouterCtrlHipReqDes
CsrWifiRouterCtrlM4ReadyToSendIndDes
CsrWifiRouterCtrlM4TransmitReqDes
CsrWifiRouterCtrlM4TransmittedIndDes
CsrWifiRouterCtrlMediaStatusReqDes
CsrWifiRouterCtrlMicFailureIndDes
CsrWifiRouterCtrlModeSetCfmDes
CsrWifiRouterCtrlModeSetReqDes
CsrWifiRouterCtrlMulticastAddressIndDes
CsrWifiRouterCtrlMulticastAddressResDes
CsrWifiRouterCtrlPeerAddCfmDes
CsrWifiRouterCtrlPeerAddReqDes
CsrWifiRouterCtrlPeerDelCfmDes
CsrWifiRouterCtrlPeerDelReqDes
CsrWifiRouterCtrlPeerUpdateCfmDes
CsrWifiRouterCtrlPeerUpdateReqDes
CsrWifiRouterCtrlPortConfigureCfmDes
CsrWifiRouterCtrlPortConfigureReqDes
CsrWifiRouterCtrlQosControlReqDes
CsrWifiRouterCtrlRawSdioDeinitialiseCfmDes
CsrWifiRouterCtrlRawSdioInitialiseCfmDes
CsrWifiRouterCtrlResumeResDes
CsrWifiRouterCtrlStaInactiveIndDes
CsrWifiRouterCtrlSuspendIndDes
CsrWifiRouterCtrlSuspendResDes
CsrWifiRouterCtrlTclasAddCfmDes
CsrWifiRouterCtrlTclasAddReqDes
CsrWifiRouterCtrlTclasDelCfmDes
CsrWifiRouterCtrlTclasDelReqDes
CsrWifiRouterCtrlTrafficClassificationReqDes
CsrWifiRouterCtrlTrafficConfigReqDes
CsrWifiRouterCtrlTrafficProtocolIndDes
CsrWifiRouterCtrlTrafficSampleIndDes
CsrWifiRouterCtrlUnexpectedFrameIndDes
CsrWifiRouterCtrlWapiRxMicCheckIndDes
CsrWifiRouterCtrlWapiRxPktReqDes
CsrWifiRouterCtrlWapiUnicastTxEncryptIndDes
CsrWifiRouterCtrlWapiUnicastTxPktReqDes
CsrWifiRouterCtrlWifiOnCfmDes
```



```
CsrWifiRouterCtrlWifiOnIndDes
CsrWifiRouterCtrlWifiOnReqDes
CsrWifiRouterCtrlWifiOnResDes
CsrWifiRouterMaPacketCancelReqDes
CsrWifiRouterMaPacketCfmDes
CsrWifiRouterMaPacketIndDes
CsrWifiRouterMaPacketReqDes
CsrWifiRouterMaPacketResDes
CsrWifiRouterMaPacketSubscribeCfmDes
CsrWifiRouterMaPacketSubscribeReqDes
CsrWifiRouterMaPacketUnsubscribeCfmDes
CsrWifiSmeAdhocConfigGetCfmDes
CsrWifiSmeAdhocConfigSetReqDes
CsrWifiSmeAssociationCompleteIndDes
CsrWifiSmeAssociationStartIndDes
CsrWifiSmeBlacklistCfmDes
CsrWifiSmeBlacklistReqDes
CsrWifiSmeCalibrationDataGetCfmDes
CsrWifiSmeCalibrationDataSetReqDes
CsrWifiSmeCcxConfigGetCfmDes
CsrWifiSmeCcxConfigSetCfmDes
CsrWifiSmeCcxConfigSetReqDes
CsrWifiSmeCloakedSsidsGetCfmDes
CsrWifiSmeCloakedSsidsSetReqDes
CsrWifiSmeCoexConfigGetCfmDes
CsrWifiSmeCoexConfigSetReqDes
CsrWifiSmeCoexInfoGetCfmDes
CsrWifiSmeConnectCfmDes
CsrWifiSmeConnectionConfigGetCfmDes
CsrWifiSmeConnectionInfoGetCfmDes
CsrWifiSmeConnectionQualityIndDes
CsrWifiSmeConnectionStatsGetCfmDes
CsrWifiSmeConnectReqDes
CsrWifiSmeCoreDumpIndDes
CsrWifiSmeDisconnectCfmDes
CsrWifiSmeErrorIndDes
CsrWifiSmeHostConfigGetCfmDes
CsrWifiSmeHostConfigSetCfmDes
CsrWifiSmeHostConfigSetReqDes
CsrWifiSmeIbssStationIndDes
CsrWifiSmeInfoIndDes
CsrWifiSmeInterfaceCapabilityGetCfmDes
CsrWifiSmeKeyCfmDes
CsrWifiSmeKeyReqDes
CsrWifiSmeLinkQualityGetCfmDes
CsrWifiSmeMediaStatusIndDes
CsrWifiSmeMibConfigGetCfmDes
CsrWifiSmeMibConfigSetReqDes
CsrWifiSmeMibGetCfmDes
CsrWifiSmeMibGetNextCfmDes
CsrWifiSmeMibGetNextReqDes
CsrWifiSmeMibGetReqDes
CsrWifiSmeMibSetReqDes
CsrWifiSmeMicFailureIndDes
CsrWifiSmeMulticastAddressCfmDes
CsrWifiSmeMulticastAddressReqDes
CsrWifiSmePacketFilterSetCfmDes
CsrWifiSmePacketFilterSetReqDes
CsrWifiSmePermanentMacAddressGetCfmDes
CsrWifiSmePmkidCandidateListIndDes
CsrWifiSmePmkidCfmDes
CsrWifiSmePmkidReqDes
CsrWifiSmePowerConfigGetCfmDes
CsrWifiSmePowerConfigSetReqDes
CsrWifiSmeRegulatoryDomainInfoGetCfmDes
CsrWifiSmeRoamCompleteIndDes
```

```
CsrWifiSmeRoamingConfigGetCfmDes
CsrWifiSmeRoamingConfigSetCfmDes
CsrWifiSmeRoamingConfigSetReqDes
CsrWifiSmeRoamStartIndDes
CsrWifiSmeScanConfigGetCfmDes
CsrWifiSmeScanConfigSetReqDes
CsrWifiSmeScanFullReqDes
CsrWifiSmeScanResultIndDes
CsrWifiSmeScanResultsGetCfmDes
CsrWifiSmeSetReqDes
CsrWifiSmeSmeCommonConfigGetCfmDes
CsrWifiSmeSmeCommonConfigSetReqDes
CsrWifiSmeSmeStaConfigGetCfmDes
CsrWifiSmeSmeStaConfigSetCfmDes
CsrWifiSmeSmeStaConfigSetReqDes
CsrWifiSmeStationMacAddressGetCfmDes
CsrWifiSmeTspecCfmDes
CsrWifiSmeTspecIndDes
CsrWifiSmeTspecReqDes
CsrWifiSmeVersionsGetCfmDes
CsrWifiSmeWifiFlightmodeReqDes
CsrWifiSmeWifiOnIndDes
CsrWifiSmeWifiOnReqDes
CsrWifiSmeWpsConfigurationReqDes
ct_timer_instance_new
ct_timer_new
cx88_core_create
cxgb_alloc_mem
cxgbi_alloc_big_mem
cxgbi_device_register
cxgbi_sock_create
cyberpro_alloc_fb_info
DAC960_DetectController
datablob_format
dca_allocate_domain
dcookie_register
ddp_make_gl
decode_device
decode_ds_addr
dell_wmi_prepare_new_keymap
device_alloc
devinfo_seq_start
_dev_list_add
dev_seq_start
dfs_pattern_detector_init
dib8000_attach
dlm_allocate_lvb
dlm_alloc_ctxt
dlm_alloc_pagevec
dma_attach
dm_alloc_md_mempools
dma_ops_domain_alloc
dma_pin_iovec_pages
dm_bio_prison_create
dm_deferred_set_create
dm_dirty_log_create
dm_tm_create_non_blocking_clone
dn_dev_alloc_ifa
dn_dev_create
dn_new_zone
do_get_edid
drm_agp_init
drm_do_get_edid
drm_gem_object_alloc
drm_master_create
drm_mm_kmalloc
```



```
drm_mode_create
drm_pci_alloc
drm_property_create
drm_property_create_blob
dsp_cmx_new_conf
dst_cow_metrics_generic
dwc2_hcd_qh_create
dwc2_hcd_urb_alloc
dwmac1000_setup
dwmac100_setup
eb_create
edac_device_alloc_ctl_info
edac_mc_alloc
edac_pci_alloc_ctl_info
efx_alloc_channel
efx_copy_channel
eg_cache_add_entry
ehci_qh_alloc
ext3_htree_create_dir_info
ext4_htree_create_dir_info
ext4_kvmalloc
ext4_kvzalloc
ext4_li_request_new
ezusb_alloc_ctx
__fa_get_part
fb_create_modedb
fb_do_probe_ddc_edid
fc_exch_mgr_add
fc_exch_mgr_alloc
fcoe_ctlr_device_add
fc_rport_create
fd_setup_write_same_buf
fib6_alloc_table
fib_trie_table
fifo_alloc
fir16_create
fl_create
flex_array_alloc
flexcop_device_kmalloc
fnic_dev_register
fq_codel_zalloc
fscache_alloc_retrieval
fs_path_alloc
fusbh200_qh_alloc
fuse_file_alloc
fw_create_instance
fwnet_pd_new
fw_node_create
gameport_allocate_port
gcov_info_dup
gcov_iter_new
genprobe_ident_chips
get_client
getdqbuf
get_id
get_indirect_ea
get_irq_table
get_rndis_device
get_rndis_request
get_scq
get_unbound_pool
get_vm_block
get_xol_area
gf128mul_init_4k_bbe
gf128mul_init_4k_lle
gf128mul_init_64k_bbe
```

```
gf128mul_init_64k_lle
gfs2_alloc_sort_buffer
gigaset_initdriver
gntdev_alloc_map
go7007_alloc
groups_alloc
gru_alloc_vma_data
gsm_alloc_mux
gsm_control_send
gsm_data_alloc
gsm_dlci_alloc
gsmi_buf_alloc
gss_alloc_context
hashbin_new
hashtab_create
hci_alloc_dev
hci_chan_create
hci_conn_add
hfs_btree_open
hfsplus_btree_open
hidinput_allocate
hid_register_field
__hostap_add_bss
hotadd_new_pgdat
hpfs_get_4sectors
hpfs_load_bitmap_directory
hpfs_load_code_page
hpfs_map_4sectors
hpi_alloc_control_cache
hsi_alloc_controller
hsi_alloc_msg
hso_create_device
hso_create_shared_int
hugepage_new_subpool
i2c_matroxfb_probe
i2c_new_device
i2o_exec_wait_alloc
i915_error_object_create_sized
i915_pages_create_for_stolen
ibmasm_new_command
ibm_slot_from_id
ib_ucm_ctx_alloc
ics5342_init
idt77252_init_est
iio_alloc_pollfunc
iio_device_alloc
iio_trigger_alloc
il4965_sta_alloc_lq
imon_init_intf0
in_cache_add_entry
inetdev_init
inet_frag_alloc
init_bch
init_list_set
init_sbd
init_temp_data
init_test_block_group
input_allocate_device
input_allocate_polled_device
intel_crtc_mode_get
intel_display_capture_error_state
intel_overlay_capture_error_state
ioat2_alloc_ring
ioat_dma_alloc_descriptor
iommu_domain_alloc
ipack_bus_register
```



```
ipath_create_mmap_info
ipath_mcast_alloc
ipath_mcast_qp_alloc
ipath_user_sdma_queue_create
ipc_alloc
ipmi_alloc_recv_msg
ipmi_alloc_smi_msg
ipoib_cm_create_tx
ipoib_mcast_alloc
ipoib_mcast_iter_init
ipoib_neigh_ctor
ipoib_path_iter_init
ipr_alloc_ucode_buffer
ip_set_alloc
ipv6_add_dev
ip_vs_lblc_new
ip_vs_lblcr_new
ip_vs_sync_buff_create
ip_vs_sync_buff_create_v0
ipw_alloc_error_log
ipwireless_hardware_create
ipwireless_network_create
ipw_rx_queue_alloc
ipxitf_alloc
ircomm_open
irda_usb_find_class_desc
iriap_open
irias_new_integer_value
irias_new_missing_value
irias_new_object
irias_new_octseq_value
irias_new_string_value
irlap_open
irlmp_copy_discoveries
irlmp_open_lsap
irlmp_register_client
irlmp_register_service
irq_alloc_generic_chip
irttp_open_tsap
iscsi_alloc_session
iscsi_boot_create_kobj
iscsi_boot_create_kset
iscsi_create_conn
iscsi_create_endpoint
iscsi_create_flashnode_conn
iscsi_create_flashnode_sess
iscsi_create_iface
iscsi_login_init_conn
iscsi_set_default_param
iscsit_alloc_portal_group
isdn_audio_adpcm_init
isdn_audio_dtmf_init
isdn_audio_silence_init
isdn_ppp_ccp_reset_alloc
isdn_ppp_ccp_reset_alloc_state
isdn_v110_open
iser_device_find_by_ib_device
iso_sched_alloc
iso_stream_alloc
it821x_firmware_command
iwl_parse_eeprom_data
iwl_parse_nvm_data
iwl_phy_db_init
iwl_sta_alloc_lq
iwl_trans_pcie_alloc
jffs2_alloc_full_dirent
```

```
journal_init_common
kcalloc
kmalloc
kmem_alloc
kobject_create
kobject_get_path
kobj_map_init
kset_create
kstruprdup
kvm_create_pic
kvm_create_pit
kzalloc
l2cap_chan_create
l2tp_session_create
lapb_create_cb
lc_create
leaf_info_new
lib80211_ccmp_init
lib80211_tkip_init
lib80211_wep_init
libipw_alloc_txb
line6_alloc_sysex_buffer
linear_conf
llc_nop_init
llc_sap_alloc
llc_shdlc_init
locate_module_kobject
lowpan_alloc_new_frame
lpddr_cmdset
lpddr_probe_chip
lpfc_alloc_fast_evt
lpfc_bsg_dma_page_alloc
lpfc_bsg_event_new
lpfc_create_vport_work_array
lpfc_els_hbq_alloc
lpfc_hba_alloc
lpfc_sli4_create_rpi_hdr
lpfc_sli4_queue_alloc
lpfc_sli4_rb_alloc
make_8023_client
make_acpi_ec
make_auth_domain_name
make_cm_node
make_EII_client
make_entry
make_slot_name
map_absent_probe
map_ram_probe
map_rom_probe
match_strdup
matroxfb_crtc2_probe
mb_cache_create
mca_bucket_alloc
mdiobus_alloc_size
md_register_thread
mei_cl_add_device
mei_cl_allocate
mei_io_cb_init
mei_me_dev_init
mem_cgroup_alloc
memstick_alloc_card
memstick_alloc_host
memtype_get_idx
merge_attr
mesh_table_alloc
mgag200_i2c_create
```



```
mgag200_ttm_tt_create
mgmt_pending_add
mgsl_allocate_device
mini_cm_listen
minstrel_alloc
minstrel_alloc_sta
minstrel_ht_alloc_sta
mISDN_register_clock
mlx4_alloc_db_pgdir
mlx4_alloc_icm
mlx4_en_add
mlx4_en_filter_alloc
mlx5_alloc_db_pgdir
mlx5_vzalloc
mmc_alloc_host
mmc_alloc_sg
mmc_test_alloc_mem
mousevsc_alloc_device
mpi_alloc
mpi_get_buffer
mppe_alloc
mpt2sas_transport_port_add
mpt3sas_transport_port_add
mptsas_expander_add
mptscsih_info
mthca_alloc_icm
mthca_alloc_icm_table
mv88e1xxx_phy_create
mv88x201x_phy_create
mvs_pci_alloc
mv_u3d_build_trb_one
mvumi_alloc_mem_resource
mvumi_create_internal_cmd
mwifiex_add_sta_entry
mwifiex_wmm_allocate_ralist_node
my3126_phy_create
nand_bch_init
nci_allocate_device
ncp_alloc_req
neigh_alloc
neigh_hash_alloc
nes_cm_alloc_core
nes_get_cqp_request
net_alloc_generic
netdev_create_hash
netlbl_secattr_cache_alloc
netlbl_unlhsh_add_iface
new_adapter
new_compound_name
_new_connect_scan_req
new_coredump_node
new_dir
new_inode_smack
new_l3_process
new_lanai_vcc
new_links
new_ncci
new_node
new_plci
new_pts_fs_info
new_tape_buffer
new_task_smack
newtframe
new_writequeue_entry
nf_bridge_alloc
nfc_allocate_device
```

```
nfc_hci_allocate_device
nfc_llc_allocate
nfc_llcp_build_sdreq_tlv
nfc_llcp_build_sdres_tlv
nfc_llcp_build_tlv
nfc_mei_phy_alloc
nfs3_alloc_createdata
nfs4_acl_new
nfs4_alloc_createdata
nfs4_alloc_lockdata
nfs4_alloc_lock_state
nfs4_alloc_open_state
nfs4_alloc_pages
nfs4_alloc_session
nfs4_alloc_state_owner
nfs4_alloc_unlockdata
nfs4_blk_decode_device
nfs4_new_slot
nfs4_opendata_alloc
nfs4_pnfs_ds_add
nfs4_pnfs_remotestr
nfs_alloc_createdata
nfs_alloc_fattr
nfs_alloc_fhandle
nfs_alloc_parsed_mount_data
nfs_alloc_seqid
nfs_alloc_server
nfs_cache_defer_req_alloc
nfsd4_cltrack_legacy_recdir
nfsd4_cltrack_legacy_topdir
nfs_readdata_alloc
nfs_writedata_alloc
n_hdlc_alloc
nilfs_segctor_new
niu_new_parent
nlm_alloc_call
nlm_alloc_host
nlmclnt_prepare_block
nlmsvc_create_block
nsm_create_handle
nvme_alloc_iod
nvme_alloc_ns
nvme_alloc_queue
nvme_bio_split
ocfs2_allocate_refcount_tree
ocfs2_alloc_quota_recovery
ocfs2_dx_dir_kmalloc_leaves
ocfs2_new_dlm_debug
ocfs2_new_path
ocfs2_xattr_bucket_new
ocrdma_init_emb_mqe
_osd_request_alloc
OS_kmalloc
oslec_create
ovs_flow_tbl_alloc
oz_alloc_urb_link
oz_elt_info_alloc
oz_ep_alloc
oz_pd_alloc
oz_tx_frame_alloc
p54_convert_db
p9_fcall_alloc
padata_alloc
padata_alloc_pd
panel_bind_key
parkbd_allocate_serio
```



```
parport_register_device
parport_register_port
path_rec_create
pch_uart_init_port
pci_alloc_bus
pci_alloc_dev
pci_alloc_host_bridge
pci_create_slot
pcie_init
pci_mmconfig_alloc
pciserial_init_ports
pci_store_saved_state
pcistub_device_alloc
pcmcia_device_add
pcmcia_make_resource
pcpu_get_vm_areas
pcrypt_alloc_instance
pfkey_sadb2xfrm_user_sec_ctx
__phonet_device_alloc
picolcd_send_and_wait
pkt_alloc_packet_data
pkt_kobj_create
pm3393_mac_create
pmcraid_alloc_sglist
pneigh_lookup
pnfs_mdsthreshold_alloc
pnp_add_card_id
pnp_add_id
pnp_alloc
pnp_alloc_card
pnp_alloc_dev
pnp_build_option
pnp_new_resource
pool_allocate
pool_alloc_page
posix_acl_alloc
pps_register_source
pri_detector_init
prism2_read_pda
__proc_create
protection_domain_alloc
psb_gtt_alloc_range
psb_intel_crtc_mode_get
psb_intel_i2c_create
psb_mmu_alloc_pd
psb_mmu_alloc_pt
psb_mmu_driver_init
pstore_ftrace_seq_start
publ_create
pvr2_context_create
pvr2_dvb_create
pvr2_eeprom_fetch
pvr2_full_eeprom_fetch
pvr2_hdw_create
pvr2_ioread_create
pvr2_std_create_enum
pvr2_stream_create
pvr2_sysfs_class_create
pvr2_sysfs_create
pvr2_v4l2_create
qcam_init
qdisc_class_hash_alloc
qib_create_mmap_info
qib_mcast_alloc
qib_mcast_qp_alloc
qib_qp_iter_init
```

```
qib_user_sdma_queue_create
qla2x00_alloc_fcport
qla2x00_alloc_work
qla4xxx_alloc_work
qla4xxx_get_new_mrb
qla84xx_get_chip
qlcnic_sriov_get_free_node_async_work
qlt_get_port_database
qp_guest_endpoint_create
qp_host_alloc_queue
qset_new_std
queue_new
quickstart_buttons_add
qxl_ring_create
r8a66597_make_td
radeon_atombios_get_lvds_info
radeon_atombios_get_primary_dac_info
radeon_atombios_get_tv_dac_info
radeon_atombios_set_dac_info
radeon_atombios_set_dig_info
radeon_bios_get_hardcoded_edid
radeon_combios_get_lvds_info
radeon_combios_get_primary_dac_info
radeon_combios_get_tv_dac_info
radeon_i2c_create
radeon_i2c_create_dp
radeon_legacy_get_ext_tmds_info
radeon_legacy_get_lvds_info_from_regs
radeon_legacy_get_tmds_info
radeon_vm_bo_add
rate_control_alloc
rate_control_pid_alloc
rate_control_pid_alloc_sta
rb_alloc
rbd_dev_create
rbd_spec_alloc
rc_allocate_device
rcu_string_strdup
rdma_create_xprt
rds_message_alloc
read_rds_samples
realloc_argv
recent_entry_init
record_old_file_extents
regcache_rbtree_node_alloc
regdom_intersect
register_8022_client
register_memory_resource
register_snap_client
__register_sysctl_paths
__register_sysctl_table
relay_create_buf
relay_open
result_init
resv_map_alloc
rfcomm_dlc_alloc
rfcomm_session_add
rfkill_alloc
__ring_buffer_alloc
ring_buffer_read_prepare
rio_alloc_net
rio_request_outb_dbell
rio_setup_device
rndis_add_response
rproc_alloc
rs_init
```



```
rtl_rate_alloc_sta
rxrpc_alloc_bundle
rxrpc_alloc_connection
rxrpc_alloc_local
rxrpc_alloc_peer
rxrpc_alloc_transport
saa7164_buffer_alloc
saa7164_buffer_alloc_user
sas_alloc_device
sas_phy_alloc
sas_port_alloc
sb_open
sch56xx_watchdog_register
scsi_alloc_sdev
scsi_bios_ptable
scsi_host_alloc
scsi_prep_async_scan
sctp_association_new
sctp_auth_create_key
sctp_auth_shkey_create
sctp_datamsg_new
sctp_endpoint_new
sctp_pack_cookie
sctp_ssnmap_new
sctp_transport_new
sd_alloc_ctl_entry
sdebug_device_create
sdev_evt_alloc
sep_queue_status_add
seq_create_client1
__seq_open_private
sesInfoAlloc
sfi_sysfs_install_table
sfq_alloc
sf_sample_new
sf_zone_new
sg_add_sfp
sid_to_key_str
simple_xattr_alloc
slhc_init
sm_create_sysfs_attributes
smi_info_alloc
smk_import_entry
smk_parse_smack
smp_chan_create
smscore_createbuffer
smtc_alloc_fb_info
snd_ctl_new
snd_emux_create_port
snd_info_create_entry
snd_midi_channel_alloc_set
snd_midi_channel_init_set
snd_pdacf_create
snd_seq_create_port
snd_seq_fifo_new
snd_seq_oss_readq_new
snd_seq_oss_timer_new
snd_seq_oss_writeq_new
snd_seq_pool_new
snd_seq_prioq_new
snd_seq_timer_new
snd_sf_new
snd_timer_instance_new
snd_usb_add_endpoint
__snd_util_memblk_new
snd_util_memhdr_new
```

```
snd_vx_create
sock_kmalloc
sparse_index_alloc
spi_alloc_device
spi_alloc_master
squashfs_cache_init
srp_add_port
srp_alloc_iu
srp_ring_alloc
srpt_alloc_ioctx
srpt_alloc_ioctx_ring
sta_info_alloc
st_allocate_request
stub_device_alloc
submit_async_request
subscr_named_msg_event
subscr_subscribe
__svc_create
synaptics_i2c_touch_create
sync_fence_alloc
sync_pt_create
sync_timeline_create
sysfs_init_inode_attrs
t1_espi_create
t1_sge_create
t1_tp_create
t4_alloc_mem
table_seq_start
tconInfoAlloc
tcpm_new
tcp_v4_save_options
td_alloc_init_desc
tg3_vpd_readblock
tifm_alloc_adapter
tipc_alloc_entry
tipc_createport
tipc_link_create
tipc_nameseq_create
tipc_subseq_alloc
tomoyo_commit_ok
tomoyo_init_log
tomoyo_print_bprm
tomoyo_print_header
tpm_bios_log_setup
tpm_register_hardware
trusted_options_alloc
__ttm_dma_alloc_page
ttm_object_device_init
ttm_object_file_init
tty_audit_buf_alloc
tty_buffer_alloc
tun_flow_create
_ubh_bread_
ucma_alloc_ctx
ucma_alloc_multicast
udf_sb_alloc_bitmap
udl_encoder_init
udl_gem_alloc_object
udl_get_edid
uf_sdio_mmc_register_pm_notifier
ulist_alloc
umc_device_create
unifi_alloc_card
usb_alloc_dev
usb_alloc_urb
usb_cache_string
```



```
usb_create_shared_hcd
usbctlx_alloc
usbhsh_ureq_alloc
usbip_alloc_iso_desc_pdu
usbvision_alloc
uvc_alloc_entity
uvesafb_prep
__uwb_beca_add
uwb_drp_ie_alloc
uwb_rc_alloc
uwb_rsv_alloc
v4l2_ctrl_new
v9fs_alloc_rdir_buf
vb2_dc_get_base_sgt
vb2_dma_sg_alloc
vb2_dma_sg_get_userptr
vb2_get_vma
vb2_vmalloc_alloc
vb2_vmalloc_get_userptr
via_aux_probe
via_new_spec
vic_provinfo_alloc
__videobuf_alloc
__videobuf_alloc_vb
videobuf_dvb_alloc_frontend
vlan_info_alloc
vlan_vid_info_alloc
vlsi_alloc_ring
vmalloc_to_sg
vmbus_device_create
vmci_handle_arr_create
vme_dma_pattern_attribute
vme_dma_pci_attribute
vme_dma_request
vme_dma_vme_attribute
vme_lm_request
vme_master_request
vme_new_dma_list
vme_slave_request
vmw_fence_manager_init
vmxnet3_copy_mc
vnic_dev_register
vq_req_alloc
vsc7326_mac_create
vxfs_getfsh
__vxge_hw_channel_allocate
vxge_os_dma_malloc
w1_alloc_dev
wakeup_source_create
wa_xfer_create_subset_sg
wiimote_create
wl1271_op_prepare_multicast
wlc_phy_shared_attach
wlc_phy_shim_attach
wm_adsp_buf_alloc
wpan_phy_alloc
wusb_dev_alloc
xbv_to_patch
xfrm_hash_alloc
xfrm_policy_alloc
xfrm_state_alloc
xhci_alloc_command
xhci_alloc_container_ctx
xhci_alloc_stream_info
xhci_ring_alloc
xhci_segment_alloc
```

```
1183 xprt_alloc
1184 xprt_dynamic_alloc_slot
1185 xt_alloc_table_info
1186 xz_dec_bcj_create
1187 xz_dec_init
1188 xz_dec_lzma2_create
1189 zbud_create_pool
1190 z_comp_alloc
1191 z_decomp_alloc
1192 zoran_setup_videocodec
1193 zram_meta_alloc
1194 zs_create_pool
```

```
**********************************************************
       0 Fri Dec 21 15:00:20 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.allocation_funcs.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
   14809 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.allocation_funcs_gfp
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 // Automatically generated by add_gfp_to_allocations.sh
  2 aarp_alloc X
  3 acpi_processor_alloc_pdc X
  4 add_device X
  5 add_missing_dev X
  6 add_one_partition X
  7 adfs_read_map X
  8 aer_alloc_rpc X
  9 afs_alloc_flat_call X
 10 afs_alloc_server X
 11 afs_vlocation_alloc X
 12 agp_alloc_bridge X
 13 agp_create_client X
 14 agp_create_controller X
 15 agp_create_memory X
 16 agp_create_user_memory X
 17 ahc_alloc X
 18 ahc_alloc_tstate X
 19 ahd_alloc X
 20 ahd_alloc_tstate X
 21 alloc_area X
 22 alloc_async X
 23 allocate_bitmap_node X
 24 allocate_request X
 25 __alloc_atm_dev X
 26 alloc_bd_holder X
 27 alloc_behind_pages X
 28 alloc_buf X
 29 alloc_buffer X
 30 alloc_cell X
 31 alloc_chunk X
 32 alloc_client X
 33 alloc_context X
 34 alloc_ctrl_packet X
 35 alloc_data_packet X
 36 alloc_dca_provider X
 37 alloc_dev X
 38 alloc_dma_iso_ctx X
 39 alloc_ebda_hpc X
 40 alloc_ep 1
 41 alloc_error_bus X
 42 alloc_fdtable X
 43 alloc_group_attrs X
 44 alloc_ioapic_entries X
 45 alloc_jh X
 46 alloc_journal_list X
 47 alloc_mpc X
 48 alloc_msi_entry X
 49 alloc_multipath X
 50 AllocNetDevice X
 51 alloc_nfs_open_context X
 52 alloc_nilfs X
 53 _alloc_path_selector X
 54 alloc_pci_dev X
 55 alloc_pcie_link_state X
 56 alloc_pcpu_chunk X
 57 __alloc_pending_request 0
 58 alloc_pgpath X
 59 alloc_pg_vec X
 60 alloc_pipe_info X
```



```
 61 alloc_pl X
 62 alloc_priority_group X
 63 alloc_rdma X
 64 alloc_read_gpt_entries X
 65 alloc_read_gpt_header X
 66 alloc_resources X
 67 alloc_rootdomain X
 68 alloc_sched_domains X
 69 alloc_scq X
 70 alloc_selector X
 71 alloc_sglist X
 72 alloc_sock_iocb X
 73 alloc_super X
 74 alloc_symbol_cache X
 75 alloc_tree X
 76 alloc_tty_driver X
 77 alloc_urbs X
 78 AllocVmbusChannel X
 79 amd_alloc_nb X
 80 aoedev_by_sysminor_m X
 81 ap_add_sta X
 82 ap_auth_make_challenge X
 83 append_internal X
 84 argv_split 0
 85 ata_port_alloc X
 86 ath_gen_timer_alloc X
 87 ath_rate_alloc_sta 2
 88 atif_add_device X
 89 atom_parse X
 90 audit_buffer_alloc 1
 91 audit_krule_to_data X
 92 audit_krule_to_rule X
 93 autofs4_init_ino X
 94 avmcard_dma_alloc X
 95 ax25_create_cb X
 96 b1_alloc_card X
 97 b43_calibrate_lo_setting X
 98 b43_generate_dyn_tssi2dbm_tab X
 99 b43legacy_generate_probe_resp X
100 b43legacy_setup_dmaring X
101 b43legacy_setup_pioqueue X
102 b43_setup_dmaring X
103 b43_setup_pioqueue_rx X
104 b43_setup_pioqueue_tx X
105 bio_alloc_map_data 2
106 bio_kmalloc 0
107 bioset_create X
108 bitmap_alloc_page X
109 __blk_queue_init_tags X
110 brd_alloc X
111 br_multicast_new_group X
112 bsd_alloc X
113 bsg_alloc_device X
114 btmrvl_add_card X
115 build_path_from_dentry X
116 cachefiles_cook_key X
117 call_usermodehelper_setup 3
118 capiminor_alloc X
119 capincci_alloc X
120 cas_page_alloc 1
121 cciss_allocate_sg_chain_blocks X
122 cdebbuf_alloc X
123 cdev_alloc X
124 cfi_read_pri X
125 cfi_staa_setup X
126 cifs_build_path_to_root X
```

```
cifs_new_fileinfo X
cifs_strndup_from_ucs X
class_compat_register X
clusterip_config_init X
cmtp_application_add X
cnic_alloc_dev X
cn_queue_alloc_callback_entry X
cn_queue_alloc_dev X
Config_FileOperation X
copy_tlv X
country_ie_2_rd X
cpia2_init_camera_struct X
cpia_register_camera X
cpqhp_slot_create X
create_a X
create_bounce_buffer X
create_driver X
create_event X
create_l2 X
create_logical_pred X
create_neighbor X
create_port X
create_pred X
create_regulator X
create_rmpp_recv X
create_serial X
create_trace_option_files X
create_unique_id X
create_uts_ns X
create_wlan X
__create_workqueue_key X
__crypto_alloc_tfm X
crypto_create_tfm X
cs46xx_dsp_spos_create X
cs5535_mfgpt_alloc_timer X
csr1212_create_csr X
csr1212_new_keyval X
ct_timer_instance_new X
ct_timer_new X
cx88_core_create X
cxgb3i_c3cn_create X
cxgb3i_ddp_make_gl 4
cxgb_alloc_mem X
cyberpro_alloc_fb_info X
cypress_buf_alloc X
DAC960_DetectController X
dca_allocate_domain X
dcookie_register X
debug_buffer_allocate X
device_alloc X
devinfo_seq_start X
dev_new X
dev_seq_start X
dlm_allocate_lvb X
dlm_allocate_rsb X
dlm_alloc_ctxt X
dlm_alloc_pagevec X
dm_alloc_md_mempools X
dma_pin_iovec_pages X
dm_dirty_log_create X
dn_dev_alloc_ifa X
dn_dev_create X
dn_new_zone X
drbd_new_device X
drm_agp_init X
drm_gem_object_alloc X
```



```
drm_get_edid X
drm_master_create X
drm_mm_kmalloc 1
drm_mode_create X
drm_pci_alloc X
drm_property_create X
drm_property_create_blob X
drm_sman_alloc X
dsp_cmx_new_conf X
edac_device_alloc_ctl_info X
edac_mc_alloc X
edac_pci_alloc_ctl_info X
edge_buf_alloc X
efx_tsoh_heap_alloc X
eg_cache_add_entry X
ehci_qh_alloc X
ext3_htree_create_dir_info X
ext4_htree_create_dir_info X
ext4_init_io_end 1
__fa_get_part 2
fb_create_modedb X
fb_do_probe_ddc_edid X
fc_exch_mgr_add X
fc_exch_mgr_alloc X
fcoe_interface_create X
fc_rport_create X
fdtv_alloc X
fib6_alloc_table X
fib_hash_table X
fl_create X
flex_array_alloc 2
flexcop_device_kmalloc X
fnic_dev_register X
fn_new_zone X
frame_new X
fscache_alloc_retrieval X
fuse_file_alloc X
fwnet_pd_new X
fw_node_create X
garp_attr_create X
gcov_info_dup X
gcov_iter_new X
genprobe_ident_chips X
getdqbuf X
get_free_de X
get_indirect_ea X
GetRndisDevice X
GetRndisRequest X
get_scq X
get_video_info X
get_vm_block X
gf128mul_init_4k_bbe X
gf128mul_init_4k_lle X
gf128mul_init_64k_bbe X
gf128mul_init_64k_lle X
gfs2_alloc_get X
gigaset_initdriver X
go7007_alloc X
groups_alloc X
gss_alloc_context X
hashbin_new X
hash_new X
hashtab_create X
hci_alloc_dev X
hci_conn_add X
hfs_btree_open X
```

```
hfsplus_btree_open X
hid_register_field X
__hostap_add_bss X
hpfs_get_4sectors X
hpfs_load_bitmap_directory X
hpfs_load_code_page X
hpfs_map_4sectors X
hpsb_alloc_host X
hpsb_alloc_packet X
hpsb_iso_common_init X
hso_create_device X
hso_create_shared_int X
i1480u_tx_create 2
__i2400m_work_setup 2
i2c_matroxfb_probe X
i2c_new_device X
i2o_exec_wait_alloc X
ibmasm_new_command X
ibm_slot_from_id X
ib_ucm_ctx_alloc X
icn_initcard X
ics5342_init X
idt77252_init_est X
ieee80211_alloc_txb 2
ieee80211_ccmp_init X
ieee80211_key_alloc X
ieee80211_tkip_init X
iio_allocate_device X
iio_allocate_interrupt X
iio_allocate_trigger X
in_cache_add_entry X
inetdev_init X
inet_frag_alloc X
init_sbd X
init_send_hfcd X
input_allocate_device X
input_allocate_polled_device X
ioat2_alloc_ring 2
ioat_dma_alloc_descriptor 1
iommu_domain_alloc X
ipc_alloc X
ipc_rcu_alloc X
ipmi_alloc_recv_msg X
ipmi_alloc_smi_msg X
ipoib_cm_create_tx X
ipoib_create_ah X
ipoib_mcast_alloc X
ipoib_mcast_iter_init X
ipoib_neigh_alloc X
ipoib_path_iter_init X
ipr_alloc_ucode_buffer X
ipv6_add_dev X
ip_vs_dest_set_insert X
ipw_alloc_error_log X
ipwireless_hardware_create X
ipwireless_network_create X
ipw_rx_queue_alloc X
ipxitf_alloc X
ircomm_open X
irda_usb_find_class_desc X
iriap_open X
irias_new_integer_value X
irias_new_missing_value X
irias_new_object X
irias_new_octseq_value X
irias_new_string_value X
```



```
irlap_open X
irlmp_copy_discoveries X
irlmp_open_lsap X
irlmp_register_client X
irlmp_register_service X
irttp_dup X
irttp_open_tsap X
iscsi_alloc_session X
iscsi_create_conn X
iscsi_create_endpoint X
isdn_audio_adpcm_init X
isdn_audio_dtmf_init X
isdn_audio_silence_init X
isdn_ppp_ccp_reset_alloc X
isdn_ppp_ccp_reset_alloc_state X
isdn_v110_open X
iser_device_find_by_ib_device X
iso_sched_alloc 1
iso_stream_alloc 0
it821x_firmware_command X
jffs2_alloc_full_dirent X
journal_init_common X
kcalloc 2
kmalloc 1
kmem_alloc X
kobject_create X
kobject_get_path 1
kobj_map_init X
kset_create X
kstruprdup 1
kvasprintf 0
kvm_create_pic X
kvm_create_pit X
kzalloc 1
lapb_create_cb X
lc_create X
lib80211_ccmp_init X
lib80211_tkip_init X
lib80211_wep_init X
libipw_alloc_txb 3
line6_alloc_sysex_buffer X
linear_conf X
llc_sap_alloc X
loop_alloc X
lpddr_cmdset X
lpddr_probe_chip X
lpfc_alloc_fast_evt X
lpfc_bsg_event_new X
lpfc_create_vport_work_array X
lpfc_els_hbq_alloc X
lpfc_hba_alloc X
lpfc_sli4_create_rpi_hdr X
lpfc_sli4_queue_alloc X
lpfc_sli4_rb_alloc X
make_8023_client X
make_acpi_ec X
make_bind_capture X
make_class_name X
make_cm_node X
make_driver_name X
make_EII_client X
make_entry X
make_resource X
make_slot_name X
map_absent_probe X
map_ram_probe X
```

```
map_rom_probe X
match_strdup X
matroxfb_crtc2_probe X
mb_cache_create X
mca_attach_bus X
mdiobus_alloc X
md_register_thread X
mem_cgroup_alloc X
memstick_alloc_card X
memstick_alloc_host X
memtype_get_idx X
mesh_table_alloc X
mgsl_allocate_device X
mini_cm_listen X
minstrel_alloc X
minstrel_alloc_sta 2
mISDN_register_clock X
mlx4_alloc_db_pgdir X
mlx4_alloc_icm X
mlx4_en_add X
mmc_alloc_host X
mppe_alloc X
mpt2sas_transport_port_add X
mptsas_expander_add X
mptscsih_info X
mthca_alloc_icm X
mthca_alloc_icm_table X
mv88e1xxx_phy_create X
mv88x201x_phy_create X
mvs_alloc_task X
mvs_pci_alloc X
my3126_phy_create X
neigh_hash_alloc X
nes_cm_alloc_core X
nes_get_cqp_request X
net_alloc_generic X
netdev_create_hash X
netfs_trans_alloc X
netlbl_unlhsh_add_iface X
new_adapter X
new_inode_smack X
new_l3_process X
new_node X
new_plci X
new_pts_fs_info X
new_tape_buffer X
new_writequeue_entry 1
nfs3_alloc_createdata X
nfs4_acl_new X
nfs4_alloc_createdata X
nfs4_alloc_lockdata X
nfs4_alloc_lock_state X
nfs4_alloc_open_state X
nfs4_alloc_session X
nfs4_alloc_state_owner X
nfs4_alloc_unlockdata X
nfs4_opendata_alloc X
nfs_alloc_parsed_mount_data X
nfs_alloc_seqid X
nfs_alloc_server X
nfs_cache_defer_req_alloc X
n_hdlc_alloc X
ni_65xx_alloc_subdevice_private X
ni_gpct_device_construct X
nilfs_segctor_new X
niu_new_parent X
```



```
nlm_alloc_call X
nlmclnt_prepare_block X
nlm_lookup_host X
nlmsvc_create_block X
nodemgr_create_node X
nodemgr_process_unit_directory X
nsm_create_handle X
ocfs2_allocate_refcount_tree X
ocfs2_alloc_quota_recovery X
ocfs2_dx_dir_kmalloc_leaves X
ocfs2_new_dlm_debug X
ocfs2_new_path X
ocfs2_xattr_bucket_new X
_osd_request_alloc 0
osd_WaitEventCreate X
oslec_create X
oti6858_buf_alloc X
p54_convert_db X
padata_alloc X
padata_alloc_pd X
panel_bind_key X
parkbd_allocate_serio X
parport_register_device X
parport_register_port X
path_rec_create X
pci_alloc_bus X
pcibios_get_irq_routing_table X
pci_create_slot X
pcie_init X
pci_mmconfig_add X
pciserial_init_ports X
pcmcia_device_add X
pcpu_get_vm_areas X
pcrypt_alloc_instance X
perf_mmap_data_alloc X
__phonet_device_alloc X
pkt_alloc_packet_data X
pkt_bio_alloc X
pkt_kobj_create X
pl2303_buf_alloc X
pm3393_mac_create X
pm8001_alloc_task X
pmcraid_alloc_sglist X
pneigh_lookup X
pnp_add_card_id X
pnp_add_id X
pnp_alloc X
pnp_alloc_card X
pnp_alloc_dev X
pnp_build_option X
pnp_new_resource X
pohmelfs_name_alloc X
pool_allocate X
pool_alloc_page 1
posix_acl_alloc 1
prism2_read_pda X
prism2_wep_init X
__proc_create X
publ_create X
pvr2_context_create X
pvr2_dvb_create X
pvr2_eeprom_fetch X
pvr2_full_eeprom_fetch X
pvr2_hdw_create X
pvr2_ioread_create X
pvr2_std_create_enum X
```

```
pvr2_stream_create X
pvr2_sysfs_class_create X
pvr2_sysfs_create X
pvr2_v4l2_create X
qcam_init X
qdisc_class_hash_alloc X
qla2x00_alloc_fcport 1
qla2x00_alloc_work X
qla84xx_get_chip X
qset_new_std 3
queue_new X
r10bio_pool_alloc 0
r1bio_pool_alloc 0
r8a66597_make_td X
radeon_atombios_get_lvds_info X
radeon_atombios_get_primary_dac_info X
radeon_atombios_get_ss_info X
radeon_atombios_get_tv_dac_info X
radeon_atombios_set_dac_info X
radeon_atombios_set_dig_info X
radeon_combios_get_lvds_info X
radeon_combios_get_primary_dac_info X
radeon_combios_get_tv_dac_info X
radeon_i2c_create X
radeon_i2c_create_dp X
radeon_legacy_get_ext_tmds_info X
radeon_legacy_get_lvds_info_from_regs X
radeon_legacy_get_tmds_info X
rate_control_alloc X
rate_control_pid_alloc X
rate_control_pid_alloc_sta 2
rdma_create_xprt X
rds_message_alloc 1
rds_rdma_prepare X
read_rds_samples X
Realloc X
realloc_argv X
recent_entry_init X
regdom_intersect X
register_8022_client X
register_snap_client X
__register_sysctl_paths X
relay_alloc_page_array X
relay_create_buf X
relay_open X
__request_region X
resv_map_alloc X
rfcomm_dlc_alloc 0
rfcomm_session_add X
rfkill_alloc X
__ring_buffer_alloc X
ring_buffer_read_start X
rndis_add_response X
rpc_alloc_iostats X
rs_init X
rxrpc_alloc_bundle 0
rxrpc_alloc_connection 0
rxrpc_alloc_local X
rxrpc_alloc_peer 1
rxrpc_alloc_transport 2
saa7164_buffer_alloc X
sas_ex_discover_end_dev X
sas_ex_discover_expander X
sas_phy_alloc X
sas_port_alloc X
savemem X
```



```
sbp2_alloc_device X
scan_behind_bridge X
scm_fp_dup X
scsi_alloc_sdev X
scsi_bios_ptable X
scsi_host_alloc X
scsi_prep_async_scan X
sctp_association_new 3
sctp_auth_create_key 1
sctp_auth_make_key_vector 3
sctp_auth_shkey_create 1
sctp_datamsg_new 0
sctp_endpoint_new 1
sctp_pack_cookie X
sctp_ssnmap_new 2
sctp_transport_new 1
scx200_create_iface X
sd_alloc_ctl_entry X
sdebug_device_create 1
sdev_evt_alloc 1
seq_create_client1 X
__seq_open_private X
serial_buf_alloc X
sesInfoAlloc X
sf_sample_new X
sf_zone_new X
sg_add_sfp X
slhc_init X
slvl_init X
smk_import_entry X
smscore_createbuffer X
smtc_alloc_fb_info X
snd_ctl_new X
snd_emux_create_port X
snd_gf1_mem_xalloc X
snd_info_create_entry X
snd_midi_channel_alloc_set X
snd_midi_channel_init_set X
snd_pdacf_create X
snd_seq_create_port X
snd_seq_fifo_new X
snd_seq_oss_readq_new X
snd_seq_oss_timer_new X
snd_seq_oss_writeq_new X
snd_seq_pool_new X
snd_seq_prioq_new X
snd_seq_timer_new X
snd_sf_new X
snd_timer_instance_new X
__snd_util_memblk_new X
snd_util_memhdr_new X
snd_vx_create X
sock_kmalloc 1
sock_kmalloc 2
spi_alloc_device X
spi_alloc_master X
squashfs_cache_init X
srp_add_port X
srp_alloc_iu 2
srp_ring_alloc X
sta_info_alloc 2
st_allocate_request X
stl_allocbrd X
stli_allocbrd X
stub_device_alloc X
submit_async_request 5
```

```
subscr_subscribe X
sv11_init X
__svc_create X
svc_setup_socket X
synaptics_i2c_touch_create X
sysfs_init_inode_attrs X
t1_espi_create X
t1_sge_create X
t1_tp_create X
table_seq_start X
tconInfoAlloc X
tcp_v4_save_options X
ti_buf_alloc X
tifm_alloc_adapter X
tipc_cltr_create X
tipc_disc_init_link_req X
tipc_link_create X
tipc_nameseq_create X
tipc_subseq_alloc X
tipc_zone_create X
tomoyo_find_or_assign_new_profile X
tomoyo_realpath_from_path X
tpm_bios_log_setup X
tpm_register_hardware X
trace_create_file_ops X
ttm_object_device_init X
ttm_object_file_init X
ttm_tt_create X
tty_audit_buf_alloc X
tty_buffer_alloc X
_ubh_bread_ X
ucma_alloc_ctx X
ucma_alloc_multicast X
udf_sb_alloc_bitmap X
umc_device_create X
usb_alloc_dev X
usb_alloc_urb 1
usb_cache_string X
usb_create_hcd X
usbctlx_alloc X
usbip_alloc_iso_desc_pdu X
usbvision_alloc X
uvc_alloc_entity X
uvesafb_prep X
uvesafb_vbe_state_save X
__uwb_beca_add X
uwb_drp_ie_alloc X
uwb_rc_alloc X
uwb_rsv_alloc X
via_new_spec X
__videobuf_alloc X
videobuf_dvb_alloc_frontend X
videocodec_attach X
vlan_group_alloc X
vlsi_alloc_ring X
vmalloc_to_sg X
vme_dma_pattern_attribute X
vme_dma_pci_attribute X
vme_dma_request X
vme_dma_vme_attribute X
vme_lm_request X
vme_master_request X
vme_new_dma_list X
vme_slave_request X
vmxnet3_copy_mc X
vnic_dev_register X
```



```
vq_req_alloc X
vsc7326_mac_create X
vxfs_getfsh X
__vxge_hw_channel_allocate X
w1_alloc_dev X
wl1271_op_prepare_multicast X
wl_new_wavepoint X
wlp_create_wssid_e X
wpan_phy_alloc X
wusb_dev_alloc X
xfrm_hash_alloc X
xfrm_policy_alloc 1
xfrm_state_alloc X
xhci_alloc_command 3
xhci_alloc_container_ctx 2
xhci_ring_alloc 3
xhci_segment_alloc 1
xt_alloc_table_info X
xv_create_pool X
z_comp_alloc X
z_decomp_alloc X
zfwMemAllocate X
zlib_init X
zoran_setup_videocodec X
```

```
**********************************************************
      32 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.allocation_funcs_gfp.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 acquire_group 2
   2 acquire_group X
```

```
**********************************************************
  103092 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.bit_shifters
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of macros used as shifters.
// generated by 'gen_bit_shifters.sh'
AB8500_AD_DATA0_OFFSET 0
AB8500_DA_DATA0_OFFSET 8
ABORT_CONN 15
ABORT_ISP_ACTIVE 3
ABORT_UPCALL 6
ABS_HAT0X 16
ABS_HAT0Y 17
ABS_HAT1X 18
ABS_HAT1Y 19
ABS_HAT2X 20
ABS_HAT2Y 21
ABS_HAT3X 22
ABS_HAT3Y 23
ABS_POS_BITS 13
ABS_PRESSURE 24
ABS_RX 3
ABS_RY 4
ABS_RZ 5
ABS_X 0
ABS_Y 1
AC97_EI_DACS_SLOT_SHIFT 4
AC97_SC_SPSR_SHIFT 12
AC97_SLOT_LFE 9
AC97_SLOT_MIC 6
AC97_SLOT_PCM_CENTER 6
AC97_SLOT_PCM_LEFT_0 7
AC97_SLOT_PCM_LEFT 3
AC97_SLOT_PCM_RIGHT_0 8
AC97_SLOT_PCM_RIGHT 4
AC97_SLOT_PCM_SLEFT 7
AC97_SLOT_PCM_SRIGHT 8
AC97_SLOT_SPDIF_LEFT1 6
AC97_SLOT_SPDIF_LEFT2 10
AC97_SLOT_SPDIF_LEFT 7
AC97_SLOT_SPDIF_RIGHT1 9
AC97_SLOT_SPDIF_RIGHT2 11
AC97_SLOT_SPDIF_RIGHT 8
AC_AMPCAP_MUTE_SHIFT 31
AC_AMPCAP_NUM_STEPS_SHIFT 8
AC_AMPCAP_OFFSET_SHIFT 0
AC_AMPCAP_STEP_SIZE_SHIFT 16
AC_AMP_SET_INDEX_SHIFT 8
AC_DEFCFG_DEVICE_SHIFT 20
AC_DEFCFG_PORT_CONN_SHIFT 30
ACK_RATIO_SHIFT 4
AC_PARAM_ECW_MAX_OFFSET 12
AC_PARAM_ECW_MIN_OFFSET 8
AC_PARAM_TXOP_OFFSET 16
ACPI_BATTERY_ALARM_PRESENT 0
ACPI_BATTERY_QUIRK_PERCENTAGE_CAPACITY 2
ACPI_BATTERY_QUIRK_THINKPAD_MAH 3
ACPI_BATTERY_XINFO_PRESENT 1
ACPI_X_SLEEP_TYPE_POSITION 2
ACT_ESTAB 3
ACT_OFLD_CONN 1
ACT_OPEN_REQ 0
ACT_OPEN_RPL 2
ACT_RETRY_INUSE 21
```



```
ACT_RETRY_NOMEM 20
AD1836_ADC_WORD_OFFSET 4
AD1836_DAC_WORD_LEN_OFFSET 3
AD193X_ADC_CHAN_SHFT 4
AD193X_DAC_CHAN_SHFT 1
AD193X_DAC_WORD_LEN_SHFT 3
AD5380_CTRL_PWR_DOWN_MODE_OFFSET 13
AD5449_CTRL_SDO_OFFSET 10
AD5755_DC_DC_FREQ_SHIFT 2
AD5755_DC_DC_MAXV 0
AD5755_DC_DC_PHASE_SHIFT 4
AD5755_SLEW_RATE_SHIFT 3
AD5755_SLEW_STEP_SIZE_SHIFT 0
AD7816_TEMP_FLOAT_OFFSET 2
AD799X_CHANNEL_SHIFT 4
AD9832_FREQ_BITS 32
AD9832_PHASE_BITS 12
AD9834_FREQ_BITS 28
AD9834_PHASE_BITS 12
ADAP_INIT 0
ADAP_SLEEP 1
ADAP_STREAMING 2
ADAPTER_STATE_GOING_DOWN 1
ADAPTER_STATE_INIT_FAILED 31
ADAPTER_STATE_LINK_DOWN 2
ADAPTER_STATE_READY 3
ADAPTER_STATE_UP 0
ADAU1373_EP_CTRL_MICBIAS1_OFFSET 4
ADAU1373_EP_CTRL_MICBIAS2_OFFSET 2
ADC_FRAC_BITS 14
ADDR_LEN 6
addr_shift 12
ADM1031_UPDATE_RATE_SHIFT 2
ADMA_MMIO_BAR 4
ADT7316_T_VALUE_FLOAT_OFFSET 2
ADT7462_PWM_CHANNEL_SHIFT 5
ADT7462_PWM_RANGE_SHIFT 4
AEL2020_GPIO_MODDET 1
AF_82XX_DUMP_READING 26
AF_82XX_FW_DUMPED 24
AF_83XX_IOCB_INTR_ON 28
AF_83XX_MBOX_INTR_ON 29
AF_83XX_NO_FW_DUMP 27
AF_8XXX_RST_OWNER 25
AF_BUILD_DDB_LIST 22
AF_EEH_BUSY 20
AF_FW_RECOVERY 19
AF_GET_CRASH_RECORD 7
AF_HA_REMOVAL 12
AF_INIT_DONE 1
AF_INTERRUPTS_ON 6
AF_INTx_ENABLED 15
AF_IRQ_ATTACHED 10
AF_LINK_UP 8
AF_LOOPBACK 9
AF_MBOX_COMMAND 2
AF_MBOX_COMMAND_DONE 3
AF_MBOX_COMMAND_NOPOLL 18
AF_MSI_ENABLED 16
AF_MSIX_ENABLED 17
AF_ONLINE 0
AF_PCI_CHANNEL_IO_PERM_FAILURE 21
AF_ST_DISCOVERY_IN_PROGRESS 4
AFS_VNODE_AUTOCELL 10
AFS_VNODE_CB_BROKEN 0
AFS_VNODE_DELETED 4
```

```
AFS_VNODE_LOCKING 6
AFS_VNODE_MODIFIED 2
AFS_VNODE_MOUNTPOINT 5
AFS_VNODE_PSEUDODIR 11
AFS_VNODE_READLOCKED 7
AFS_VNODE_UNSET 1
AFS_VNODE_WRITELOCKED 8
AFS_VNODE_ZAP_DATA 3
AGP_FF_ALLOW_CLIENT 0
AGP_FF_ALLOW_CONTROLLER 1
AGP_FF_IS_CLIENT 2
AGP_FF_IS_CONTROLLER 3
AGP_FF_IS_VALID 4
AHCI_PCI_BAR 5
AIC32X4_DOSRMSB_SHIFT 4
AIC32X4_PLLJ_SHIFT 6
ALARM_ENABLE_SHIFT 7
ALI_AC97_GPIO_DATA_SHIFT 16
_ALLOC_dirid_groups 10
_ALLOC_packing_groups 12
_ALLOC_skip_busy 5
ALSA_CAPTURE_OPEN 2
ALSA_CAPTURE_RUNNING 4
ALSA_PLAYBACK_OPEN 3
ALSA_PLAYBACK_RUNNING 5
ALT_NAK_OUT_PACKETS 7
ALX_DEV_CTRL_MAXRRS_MIN 2
ALX_DMA_RCHNL_SEL_SHIFT 26
ALX_DMA_RDLY_CNT_SHIFT 11
ALX_DMA_RORDER_MODE_SHIFT 0
ALX_DMA_RREQ_BLEN_SHIFT 4
ALX_DMA_WDLY_CNT_SHIFT 16
ALX_HQTPD_Q1_NUMPREF_SHIFT 0
ALX_HQTPD_Q2_NUMPREF_SHIFT 4
ALX_HQTPD_Q3_NUMPREF_SHIFT 8
ALX_IRQ_MODU_TIMER1_SHIFT 0
ALX_MAC_CTRL_PRMBLEN_SHIFT 10
ALX_MDIO_CLK_SEL_SHIFT 24
ALX_MDIO_DATA_SHIFT 0
ALX_MDIO_EXTN_DEVAD_SHIFT 16
ALX_MDIO_EXTN_REG_SHIFT 0
ALX_MDIO_REG_SHIFT 16
ALX_MSI_RETRANS_TM_SHIFT 0
ALX_RXQ0_IDT_TBL_SIZE_SHIFT 8
ALX_RXQ0_NUM_RFD_PREF_SHIFT 20
ALX_RXQ0_RSS_MODE_SHIFT 26
ALX_RXQ2_RXF_XOFF_THRESH_SHIFT 16
ALX_RXQ2_RXF_XON_THRESH_SHIFT 0
ALX_TXQ0_TPD_BURSTPREF_SHIFT 0
ALX_TXQ0_TXF_BURST_PREF_SHIFT 16
ALX_WRR_PRI0_SHIFT 0
ALX_WRR_PRI1_SHIFT 8
ALX_WRR_PRI2_SHIFT 16
ALX_WRR_PRI3_SHIFT 24
ALX_WRR_PRI_SHIFT 29
AMP_VAL_IDX_SHIFT 19
ANALOG_FUZZ_BITS 2
ANAPARAM_PWR0_SHIFT 28
ANAPARAM_PWR1_SHIFT 20
ANAPARAM_TXDACOFF_SHIFT 27
APDS990X_APERS_SHIFT 0
APDS990X_PPERS_SHIFT 4
APERFMPERF_PRESENT 0
AR5416_EEPROM_S 2
AR5523_CONNECTED 2
AR5523_HW_UP 0
```



```
AR5523_USB_DISCONNECTED 1
AR5K_BSS_ID1_AID_S 16
AR5K_EEPROM_SIZE_ENDLOC_SHIFT 12
AR5K_IFS0_DIFS_S 11
AR5K_IFS1_CS_EN_S 26
AR5K_NODCU_RETRY_LMT_CW_MIN_S 20
AR5K_RSSI_THR_BMISS_S 8
AR_ANT_DIV_CTRL_ALL_S 25
AR_BufLen_S 16
AR_CtrlStat_S 14
AR_DescId_S 16
AR_FAST_DIV_ENABLE_S 13
ARITH_SHIFT 8
ARIZONA_AIF1_RATE_SHIFT 11
ARIZONA_AIF1TX_WL_SHIFT 8
ARIZONA_CABLE_HEADPHONE 2
ARIZONA_CABLE_MICROPHONE 1
ARIZONA_FLL1_CLK_REF_DIV_SHIFT 6
ARIZONA_FLL1_CLK_REF_SRC_SHIFT 0
ARIZONA_FLL1_FRATIO_SHIFT 8
ARIZONA_FLL1_GAIN_SHIFT 2
ARIZONA_FLL1_OUTDIV_SHIFT 1
ARIZONA_HAP_CTRL_SHIFT 2
ARIZONA_HP_IMPEDANCE_RANGE_SHIFT 9
ARIZONA_IN1_DMIC_SUP_SHIFT 11
ARIZONA_IN1_MODE_SHIFT 9
ARIZONA_LDO1_VSEL_SHIFT 5
ARIZONA_MICD_BIAS_STARTTIME_SHIFT 12
ARIZONA_MICD_DBTIME_SHIFT 1
ARIZONA_MICD_RATE_SHIFT 8
ARIZONA_OPCLK_DIV_SHIFT 3
ARIZONA_SYSCLK_FREQ_SHIFT 8
ARIZONA_SYSCLK_SRC_SHIFT 0
AR_PHY_9285_ANT_DIV_ALT_LNACONF_S 25
AR_PHY_9285_ANT_DIV_MAIN_LNACONF_S 27
AR_PHY_9285_FAST_DIV_BIAS_S 9
AR_PHY_ANT_DIV_ALT_GAINTB_S 29
AR_PHY_ANT_DIV_ALT_LNACONF_S 25
AR_PHY_ANT_DIV_LNADIV_S 24
AR_PHY_ANT_DIV_MAIN_GAINTB_S 30
AR_PHY_ANT_DIV_MAIN_LNACONF_S 27
AR_PHY_ANT_FAST_DIV_BIAS_S 9
AR_PHY_ANT_SW_RX_PROT_S 23
AR_PHY_SFCORR_SPUR_SUBCHNL_SD_S 28
AR_TxQcuNum_S 8
AR_TxRxDesc_S 15
AS_EIO 25
AS_ENOSPC 26
AS_PUSH_SHIFT 62
ASYNCB_CHECK_CD 25
ASYNCB_CLOSING 27
ASYNCB_CTS_FLOW 26
ASYNCB_INITIALIZED 31
ASYNCB_SUSPENDED 30
ATA_SHIFT_MWDMA 7
ATA_SHIFT_PIO 0
ATA_SHIFT_UDMA 12
__AT_DOWN 3
ATH10K_PCI_FEATURE_HW_1_0_WARKAROUND 1
ATH10K_PCI_FEATURE_MSI_X 0
ATH9K_NUM_TX_QUEUES 10
ATH9K_TXDESC_PAPRD_S 16
ATH_STAT_STARTED 3
ATI_REG_CMD_SPDF_THRESHOLD_SHIFT 6
ATI_REG_MODEM_OUT1_DMA_THRESHOLD_SHIFT 16
ATI_REG_OUT_DMA_THRESHOLD_SHIFT 11
```

```
ATI_REG_PHYS_OUT_ADDR_SHIFT 9
ATI_REG_PHYS_OUT_DATA_SHIFT 16
ATL1C_WORK_EVENT_LINK_CHANGE 1
ATL1C_WORK_EVENT_RESET 0
__ATL2_DOWN 2
ATM_DF_REMOVED 0
ATMEL_AT45DB0X1B_PAGE_POS 9
ATM_HDR_GFC_SHIFT 28
ATM_HDR_PTI_SHIFT 1
ATM_HDR_VCI_SHIFT 4
ATM_HDR_VPI_SHIFT 20
ATM_VF_ADDR 0
ATM_VF_BOUND 4
ATM_VF_CLOSE 11
ATM_VF_HASQOS 6
ATM_VF_HASSAP 10
ATM_VF_IS_CLIP 13
ATM_VF_LISTEN 7
ATM_VF_META 8
ATM_VF_PARTIAL 2
ATM_VF_READY 1
ATM_VF_REGIS 3
ATM_VF_RELEASED 5
ATM_VF_SESSION 9
ATM_VF_WAITING 12
ATOM_PPLL0 2
ATOM_PPLL1 0
ATOM_PPLL2 1
ATOM_PPLL_SS_AMOUNT_V2_NFRAC_SHIFT 8
ATOM_S2_CURRENT_BL_LEVEL_SHIFT 8
ATP867X_BAR_IOBASE 0
ATP867X_IO_DMAMODE_MSTR_SHIFT 0
ATP867X_IO_DMAMODE_SLAVE_SHIFT 4
ATP867X_IO_PIOSPD_ACTIVE_SHIFT 4
ATP867X_IO_PIOSPD_RECOVER_SHIFT 0
AutoDetected 4
AUTOVALIDATE 5
AXF_ERROR 2
AXF_ESCAPE 1
AXF_INUSE 0
B1_HDLC 0
B43_DMA32_DCTL_ADDREXT_SHIFT 16
B43_DMA32_RXADDREXT_SHIFT 16
B43_DMA32_RXFROFF_SHIFT 1
B43_DMA32_TXADDREXT_SHIFT 16
B43_DMA64_DCTL1_ADDREXT_SHIFT 16
B43_DMA64_RXADDREXT_SHIFT 16
B43_DMA64_RXFROFF_SHIFT 1
B43_DMA64_TXADDREXT_SHIFT 16
B43legacy_DMA32_DCTL_ADDREXT_SHIFT 16
B43legacy_DMA32_RXADDREXT_SHIFT 16
B43legacy_DMA32_RXFROFF_SHIFT 1
B43legacy_DMA32_TXADDREXT_SHIFT 16
B43legacy_TX4_MAC_KEYALG_SHIFT 16
B43legacy_TX4_MAC_KEYIDX_SHIFT 20
B43_NPHY_BPHY_CTL3_SCALE_SHIFT 0
B43_NPHY_C1_INITGAIN_HPVGA2_SHIFT 7
B43_NPHY_C1_MINGAIN_SHIFT 0
B43_NPHY_C2_INITGAIN_HPVGA2_SHIFT 7
B43_NPHY_C2_MINGAIN_SHIFT 0
B43_NPHY_OVER_DGAIN_CCKDGECV_SHIFT 8
B43_NPHY_RFCTL_CMD_CORESEL_SHIFT 3
B43_NPHY_RFSEQCA_RXDIS_SHIFT 12
B43_NPHY_RFSEQCA_RXEN_SHIFT 4
B43_NPHY_TXPCTL_ITSSI_0_SHIFT 0
B43_NPHY_TXPCTL_ITSSI_1_SHIFT 8
```



```
B43_NPHY_TXPCTL_N_NPTIL2_SHIFT 8
B43_NPHY_TXPCTL_N_TSSID_SHIFT 0
B43_NPHY_TXPCTL_TPWR_0_SHIFT 0
B43_NPHY_TXPCTL_TPWR_1_SHIFT 8
B43_PHY_BBANDCFG_RXANT_SHIFT 7
B43_PHY_HT_TXPCTL_IDLE_TSSI2_C3_SHIFT 0
B43_PHY_HT_TXPCTL_IDLE_TSSI_C1_SHIFT 0
B43_PHY_HT_TXPCTL_IDLE_TSSI_C2_SHIFT 8
B43_PHY_HT_TXPCTL_N_NPTIL2_SHIFT 8
B43_PHY_HT_TXPCTL_TARG_PWR2_C3_SHIFT 0
B43_PHY_HT_TXPCTL_TARG_PWR_C1_SHIFT 0
B43_PHY_HT_TXPCTL_TARG_PWR_C2_SHIFT 8
B43_TXH_MAC_KEYALG_SHIFT 16
B43_TXH_MAC_KEYIDX_SHIFT 20
B577XX_DOORBELL_HDR_CONN_TYPE_SHIFT 4
B577XX_DOORBELL_HDR_DB_TYPE_SHIFT 1
B577XX_DOORBELL_HDR_RX_SHIFT 0
B577XX_FCOE_RX_DOORBELL_NEGATIVE_ARM_SHIFT 0
B577XX_FCOE_RX_DOORBELL_OPCODE_SHIFT 5
BBSHIFT 9
BCD_EN_SHIFT 0
BC_FLG_ACTIV 2
BC_FLG_BUSY 3
BCMA_CC_PMU15_PLL_PC0_FREQTGT_SHIFT 2
BCMA_CC_PMU1_PLL0_PC0_P1DIV_SHIFT 20
BCMA_CC_PMU1_PLL0_PC2_NDIV_INT_SHIFT 20
BCMA_CC_PMU_CTL_RES_SHIFT 13
BCMA_CORE_PCI_MDIODATA_DEVADDR_SHF 23
BCMA_CORE_PCI_MDIODATA_DEVADDR_SHF_OLD 22
BCMA_CORE_PCI_MDIODATA_REGADDR_SHF 18
BCMA_CORE_PCI_MDIODATA_REGADDR_SHF_OLD 18
BCMA_CORE_PCI_SPROM_PI_SHIFT 12
BCN_TCFG_CW_SHIFT 8
BCN_TCFG_IFS 0
BDC_FLAG_VER_SHIFT 4
BDI_async_congested 1
BDINFO_FLAGS_MAXLEN_SHIFT 16
BDI_registered 3
B_DIRTY 2
BDI_sync_congested 2
BDI_writeback_running 4
BEACON_BLINK_NEEDED 11
BFI_MSIX_CT_MAX 9
BFUSB_TX_WAKEUP 2
BH_Delay 9
BH_Dirty 1
BH_JWrite 17
BH_Lock 2
BH_Mapped 5
BH_New 6
BH_NILFS_Node 17
BH_Pinned 16
BH_Quiet 13
BH_Uptodate 0
BIAS_MOD_LEVEL_SHIFT 8
BICTCP_HZ 10
BIO_BOUNCED 5
BIO_CLONED 4
BIO_EOF 2
BIO_EOPNOTSUPP 7
BIO_MAPPED_INTEGRITY 11
BIO_NULL_MAPPED 8
BIO_OWNS_VEC 13
BIO_QUIET 10
BIO_RESET_BITS 13
BIO_SEG_VALID 3
```

```
BIO_SNAP_STABLE 12
BIO_UPTODATE 0
BIO_USER_MAPPED 6
BIT_AGGREGATION 3
BITMAP_HOSTENDIAN 15
BITMAP_IO 9
BITMAP_STALE 1
BITMAP_WRITE_ERROR 2
BIT_MULTI_CS 0
BIT_QOS 2
BIT_WIMAX 1
BLOCKACKPARAM_WINSIZE_POS 6
Blocked 5
BlockedBadBlocks 8
BLOCK_FREED 2
BLOCK_HASH_SHIFT 16
BLOCK_NEEDS_FLUSH 4
BLUE_IS_PULSE 4
BLUE_LED 0
BLUE_PULSE_LED 1
BME_LOCKED 1
BME_NO_WRITES 0
BM_PAGE_HINT_WRITEOUT 27
BM_PAGE_IO_ERROR 30
BM_PAGE_LAZY_WRITEOUT 28
BM_PAGE_NEED_WRITEOUT 29
BM_RCTL_MO_SHIFT 3
BNAD_RF_CEE_RUNNING 0
BNAD_RF_DIM_TIMER_RUNNING 4
BNAD_RF_MBOX_IRQ_DISABLED 2
BNAD_RF_NETDEV_REGISTERED 3
BNAD_RXQ_POST_OK 1
BNAD_RXQ_STARTED 0
BNAD_TXQ_TX_STARTED 1
BNX2FC_CNIC_REGISTERED 1
BNX2FC_CTLR_INIT_DONE 0
BNX2FC_FLAG_CMD_LOST 12
BNX2FC_FLAG_CTX_ALLOC_FAILURE 6
BNX2FC_FLAG_DESTROY_CMPL 1
BNX2FC_FLAG_DESTROYED 4
BNX2FC_FLAG_DISABLED 3
BNX2FC_FLAG_DISABLE_FAILED 9
BNX2FC_FLAG_EH_ABORT 8
BNX2FC_FLAG_ELS_TIMEOUT 11
BNX2FC_FLAG_ENABLED 10
BNX2FC_FLAG_FW_INIT_DONE 0
BNX2FC_FLAG_IO_CLEANUP 6
BNX2FC_FLAG_IO_COMPL 9
BNX2FC_FLAG_ISSUE_ABTS 2
BNX2FC_FLAG_ISSUE_RRQ 1
BNX2FC_FLAG_OFFLOADED 2
BNX2FC_FLAG_OFLD_REQ_CMPL 5
BNX2FC_FLAG_RETIRE_OXID 7
BNX2FC_FLAG_SESSION_READY 1
BNX2FC_FLAG_SRR_SENT 13
BNX2FC_FLAG_TM_COMPL 4
BNX2FC_FLAG_TM_TIMEOUT 5
BNX2FC_FLAG_UPLD_REQ_COMPL 7
BNX2I_CNIC_REGISTERED 1
BNX2I_NX2_DEV_5706 0
BNX2I_NX2_DEV_5708 1
BNX2I_NX2_DEV_5709 2
BNX2I_NX2_DEV_57710 3
BNX2_L2CTX_L2_STATUSB_NUM_SHIFT 24
BNX2_PCICFG_INT_ACK_CMD_INT_NUM_SHIFT 24
BNX2X_AFEX_FCOE_Q_UPDATE_PENDING 12
```



```
BNX2X_AFEX_PENDING_VIFSET_MCP_ACK 13
BNX2X_DB_SHIFT 7
BNX2X_DONT_CONSUME_CAM_CREDIT 4
BNX2X_ETH_MAC 1
BNX2X_FILTER_RX_MODE_PENDING 3
BNX2X_FILTER_RX_MODE_SCHED 4
BNX2X_PRI_FLAG_FCOE 1
BNX2X_PRI_FLAG_ISCSI 0
BNX2X_PRI_FLAG_STORAGE 2
BNX2X_Q_TYPE_HAS_RX 0
BNX2X_Q_TYPE_HAS_TX 1
BNX2X_SP_RTNL_AFEX_F_UPDATE 3
BNX2X_SP_RTNL_ENABLE_SRIOV 4
BNX2X_SP_RTNL_FAN_FAILURE 2
BNX2X_SP_RTNL_HYPERVISOR_VLAN 8
BNX2X_SP_RTNL_TX_TIMEOUT 1
BNX2X_SP_RTNL_VFPF_CHANNEL_DOWN 6
BNX2X_SP_RTNL_VFPF_MCAST 5
BNX2X_SP_RTNL_VFPF_STORM_RX_MODE 7
BNX2X_VF_CID_WND 0
BOOST_DPM_LEVEL 7
BPP_TUNED18_SHIFT_8411 4
BP_VECTOR 3
BRCMF_FWS_FIFO_AC_BE 1
BRCMF_P2P_STATUS_ACTION_TX_COMPLETED 6
BRCMF_P2P_STATUS_ACTION_TX_NOACK 7
BRCMF_P2P_STATUS_DISCOVER_LISTEN 9
BRCMF_P2P_STATUS_ENABLED 0
BRCMF_P2P_STATUS_FINDING_COMMON_CHANNEL 13
BRCMF_P2P_STATUS_GO_NEG_PHASE 8
BRCMF_P2P_STATUS_SENDING_ACT_FRAME 10
BRCMF_P2P_STATUS_WAITING_NEXT_ACT_FRAME 12
BRCMF_P2P_STATUS_WAITING_NEXT_AF_LISTEN 11
BRCMF_SCAN_STATUS_ABORT 1
BRCMF_SCAN_STATUS_BUSY 0
BRCMF_SCAN_STATUS_SUPPRESS 2
BRCMF_VIF_STATUS_AP_CREATED 5
BRCMF_VIF_STATUS_AP_CREATING 4
BRCMF_VIF_STATUS_CONNECTED 2
BRCMF_VIF_STATUS_CONNECTING 1
BRCMF_VIF_STATUS_READY 0
B_READING 0
BR_PORT_BITS 10
B_RS_H_DONE 20
BSG_F_BLOCK 1
BS_READABLE 1
BS_WRITABLE 2
BTN_A 304
BTN_DEAD 303
BTN_EXTRA 276
BTN_LEFT 272
BTN_MIDDLE 274
BTN_RIGHT 273
BTN_SIDE 275
BTN_TOOL_DOUBLETAP 333
BTN_TOOL_FINGER 325
BTN_TOOL_QUADTAP 335
BTN_TOOL_TRIPLETAP 334
BTN_TOUCH 330
BT_OP_PRIORITY_DETECTED 0
BT_OP_SCAN 1
BTRFS_FS_STATE_ERROR 0
BTRFS_FS_STATE_REMOUNTING 1
BTRFS_INODE_COPY_EVERYTHING 8
BTRFS_INODE_DELALLOC_META_RESERVED 4
BTRFS_INODE_DUMMY 2
```

```
BTRFS_INODE_HAS_ASYNC_EXTENT 6
BTRFS_INODE_HAS_ORPHAN_ITEM 5
BTRFS_INODE_IN_DEFRAG 3
BTRFS_INODE_IN_DELALLOC_LIST 9
BTRFS_INODE_NEEDS_FULL_SYNC 7
BTRFS_INODE_ORDERED_DATA_CLOSE 0
BTRFS_ORDERED_COMPLETE 1
BTRFS_ORDERED_DIRECT 5
BTRFS_ORDERED_IOERR 6
BTRFS_ORDERED_UPDATED_ISIZE 7
BTRFS_STRIPE_HASH_TABLE_BITS 11
BTR_SJW_SHIFT 6
BTR_TSEG1_SHIFT 8
BTR_TSEG2_SHIFT 12
BT_SK_DEFER_SETUP 0
BT_SK_SUSPEND 1
BTUSB_BULK_RUNNING 1
BTUSB_DID_ISO_RESUME 4
BTUSB_SUSPENDING 3
BUFFER_EMPTY 6
BUFFER_FLUSH 7
BUFFER_FULL 7
BUFFERING_RX 2
BUFF_FLAGS_OFFSET 24
BurstLenShift 8
BUS_CNTL1_MOBILE_PLATFORM_SEL_SHIFT 26
BUS_WIDTH 0
BUSWIDTH 1
BYTE_SHIFT 3
CACHE_CLEANED 3
CACHEFILES_DEAD 1
CACHEFILES_READY 0
CACHE_NEGATIVE 1
CACHE_SET_STOPPING_2 2
CACHE_SET_UNREGISTERING 0
CACHE_VALID 0
CALIB_FRAC_BITS 10
CALLBACK_PENDING 11
CALShift 14
CAM_CTRL_INDEX_SHIFT 16
CAM_OUT_CQ_ID_SHIFT 5
CAM_OUT_FUNC_SHIFT 2
CANCEL_DC_I_SHIFT 5
CANCEL_DC_Q_SHIFT 0
CAN_SFF_ID_BITS 11
CAPTURE_URB_COMPLETED 6
CARD_HAS_ACTIVITY_LED 6
CARD_HAS_PCCARD_ID 4
CARD_HAS_POWER_LED 5
CARD_READY 1
CARL9170_TX_SUPER_AMPDU_DENSITY_S 0
CARL9170_TX_SUPER_AMPDU_FACTOR_S 3
CARL9170_TX_SUPER_RI_ERP_PROT_S 3
CB710_MMC_CMD_CODE_SHIFT 8
CDC_DCMD_ID_SHIFT 16
CDC_DCMD_IF_SHIFT 12
CEPH_BLOCK_SHIFT 22
CF_CLOSE 6
CF_CONFIG_NEEDED 1
CF_CONFIG_NEEDED 4
CF_CONNECTED 1
CF_DMA_ACTIVE 3
CFG_CHARGE_CURRENT_FCC_SHIFT 5
CFG_CHARGE_CURRENT_PCC_SHIFT 3
CFG_CURRENT_LIMIT_DC_SHIFT 4
CFG_FLOAT_VOLTAGE_THRESHOLD_SHIFT 6
```



```
CFG_OTG_CC_COMPENSATION_SHIFT 6
CFG_OTG_TEMP_THRESHOLD_SHIFT 4
CFG_Q_SHIFT 8
CFGR_BLV_SHIFT 3
CFG_TEMP_LIMIT_HARD_COLD_SHIFT 6
CFG_TEMP_LIMIT_HARD_HOT_SHIFT 4
CFG_TEMP_LIMIT_SOFT_COLD_SHIFT 2
CFG_TEMP_LIMIT_SOFT_HOT_SHIFT 0
CFG_THERM_SOFT_COLD_COMPENSATION_SHIFT 2
CFG_THERM_SOFT_HOT_COMPENSATION_SHIFT 0
CFHSI_AWAKE 3
CFHSI_FLUSH_FIFO 6
CFHSI_SHUTDOWN 5
CFHSI_WAKE_DOWN_ACK 2
CFHSI_WAKE_UP_ACK 1
CF_IS_OTHERCON 5
CFPREP_CBI_SHIFT 6
CFQ_SERVICE_SHIFT 12
CF_SG_RESTART 6
CF_SINGLE_BUFFER 5
CF_WRITE_PENDING 2
CGRP_CPUSET_CLONE_CHILDREN 3
CGRP_DEAD 0
CGRP_NOTIFY_ON_RELEASE 2
CGRP_RELEASABLE 1
CH7017_LOOP_FILTER_SHIFT 5
CH7017_LVDS_PLL_FEED_BACK_DIVIDER_SHIFT 4
CH7017_LVDS_PLL_FEED_FORWARD_DIVIDER_SHIFT 0
CH7017_LVDS_PLL_POST_SCALE_DIV_SHIFT 0
CH7017_LVDS_PLL_VCO_SHIFT 4
CH7017_PHASE_DETECTOR_SHIFT 0
CHANNEL_ABORT 2
CHANNEL_CLEAR_INTERRUPT 3
CHANNEL_DONE 4
CHANNEL_DOWN 2
CHANNEL_ENABLE 0
CHANNEL_START 1
CHIP_RESET 3
CHIP_SELECT_BIT12 12
CHUNK_SHIFT 6
CISTPL_POWER_VNOM 0
CLEAN_LIST_BUSY_BIT 0
CLEAN_SHUTDOWN 0
CLEAR_BSSFILTER_ON_BEACON 5
CLEAR_CONTROL_STATUS_PHASE_HANDSHAKE 3
CLEAR_ENDPOINT_HALT 0
CLEAR_ENDPOINT_TOGGLE 1
CLEAR_EP_FORCE_CRC_ERROR 5
CLEAR_EP_HIDE_STATUS_PHASE 6
CLEAR_INTERRUPT_MODE 4
CLEAR_NAK_OUT_PACKETS 7
CLEAR_NAK_OUT_PACKETS_MODE 2
CLK_27M_MCLK_SHIFT 3
CLK312_EN_LBN 3
CLK_32K_SR_SHIFT 4
CLKDIV_IN_SHIFT 6
CLKOUT_SL_SHIFT 4
CLK_PWRMGT_CNTL__ACTIVE_HILO_LAT__SHIFT 13
CLK_RATIO_SHIFT 1
CLK_SPEED_SHIFT 5
CLOCK_SELECT_SHIFT 4
CLOSE_SENT 3
CLOSE_UPCALL 8
CL_SEL_POS 1
CL_ST_CHG_FAIL 4
CL_ST_CHG_SUCCESS 3
```

```
CMA_OPTION_AFONLY 0
CM_ASFC_SHIFT 10
CM_CH0FMT_SHIFT 0
CM_CH1FMT_SHIFT 2
CMDQ_STAT_LAST_PKT_DB 2
CMDQ_STAT_RUNNING 1
CM_DSFC_SHIFT 13
CMTP_LOOPBACK 0
CNF1_SJW_SHIFT 6
CNF2_PS1_SHIFT 3
CNIC_F_BNX2_CLASS 3
CNIC_F_BNX2X_CLASS 4
CNIC_F_CNIC_UP 1
CNIC_LCL_FL_KWQ_INIT 0
CNIC_LCL_FL_L2_WAIT 1
CNIC_LCL_FL_RINGS_INITED 2
CNIC_LCL_FL_STOP_ISCSI 4
COMMANDII_RANGE_SHIFT 0
COMMANDII_RESOLUTION_SHIFT 2
COMMANDII_SCHEME_SHIFT 7
COMMMAND1_OPMODE_SHIFT 5
CONF_CONNECT_PEND 5
CONF_EWS_RECV 8
CONFIG_CHANNEL_HT40 6
CONFIG_HT_DISABLED 8
CONFIG_POWERSAVING 7
CONFIG_QOS_DISABLED 9
CONFIGURE_PROX_SLP_SH 4
CONFIG_WIZNET_BUS_SHIFT 0
CONF_INPUT_DONE 1
CONF_LOC_CONF_PEND 9
CONF_MODE_DONE 4
CONF_MTU_DONE 3
CONF_NOT_COMPLETE 11
CONF_OUTPUT_DONE 2
CONF_RECV_NO_FCS 6
CONF_REM_CONF_PEND 10
CONF_REQ_SENT 0
CONF_STATE2_DEVICE 7
CONN_DRY_RUN 8
CONNECTED 0
CONNECT_PEND 1
CONN_LOCAL_BUSY 5
CONN_REJ_ACT 6
CONN_REMOTE_BUSY 4
CONNREQ_UPCALL 14
CONN_RNR_SENT 8
CONN_RPL_UPCALL 19
CONN_SEND_FBIT 7
CONN_SREJ_ACT 2
CONN_WD_ST_CHG_FAIL 7
CONN_WD_ST_CHG_OKAY 6
CONN_WD_ST_CHG_REQ 5
CONSIDER_RESYNC 6
CONTROL_INV_TIMEOUT 5
CONTROL_SHIFT 6
CONTROL_STATUS_INTERRUPT 1
CONTROL_STATUS_INTERRUPT 6
CONTROL_STATUS_PHASE_HANDSHAKE 3
COOKIEBITS 24
COUNTER_SHIFT 16
CPHY_SATA_DPLL_SHIFT 8
CPL_RX_DDP_STATUS_DCRC_SHIFT 21
CPL_RX_DDP_STATUS_DDP_SHIFT 16
CPL_RX_DDP_STATUS_HCRC_SHIFT 20
CPL_RX_DDP_STATUS_PAD_SHIFT 19
```



```
cpuset_subsys_id 0
CRASHED_PRIMARY 5
CR_MAXPEXP 4
CRQB_CMD_ADDR_SHIFT 8
CRQB_HOSTQ_SHIFT 17
CRQB_PMP_SHIFT 12
CRQB_TAG_SHIFT 1
CRVML_CLOCK_SHIFT 8
CS2000_LOCK_CLK_SHIFT 1
CS2000_R_SEL_SHIFT 3
CS42L52_BEEP_RATE_SHIFT 4
CS42L52_CHARGE_PUMP_SHIFT 4
CS42L52_MIC_CTL_MIC_SEL_SHIFT 6
CS42L52_MIC_CTL_TYPE_SHIFT 5
CS_ONLINE 0
CS_SCHED_LOAD_BALANCE 5
CS_SPREAD_PAGE 6
CS_SPREAD_SLAB 7
CTL_A_GAIN_SHIFT 28
CTL_A_SEL_SHIFT 24
CTL_DA_SDR_SHIFT 8
CTL_SLEW_SHIFT 4
CT_PAGE_SHIFT 12
CTRL_BITPOS_A 30
CTRL_BITPOS_DESCLIMIT 18
CTRL_BITPOS_FIFOINDEXMASK 4
CTRL_BITPOS_G 31
CTRL_BITPOS_L2SZ 0
CTX_FL_CID_ERROR 2
CTX_FL_DELETE_WAIT 1
CTX_FL_OFFLD_START 0
CURSOR_A_FIFO_WM_SHIFT 8
CURSOR_B_FIFO_WM1_SHIFT 8
CURSOR_B_FIFO_WM_SHIFT 16
CURSOR_FIFO_SR_WM1_SHIFT 0
CURSOR_X_SHIFT 0
CURSOR_Y_SHIFT 16
CX18_F_I_EOS 4
CX18_F_I_FAILED 22
CX18_F_I_LOADED_FW 0
CX18_F_I_RADIO_USER 5
CX18_F_M_NEED_SWAP 0
CX18_F_S_APPL_IO 8
CX18_F_S_CLAIMED 3
CX18_F_S_INTERNAL_USE 5
CX18_F_S_STOPPING 9
CX18_F_S_STREAMING 4
CX20442_AGC 4
CX20442_MIC 2
CX20442_SPKOUT 3
CX20442_TELIN 0
CX20442_TELOUT 1
CX23885_IR_RX_END_OF_RX_DETECTED 1
CX23885_IR_RX_FIFO_SERVICE_REQ 0
CX23885_IR_RX_HW_FIFO_OVERRUN 2
CX23885_IR_RX_SW_FIFO_OVERRUN 3
CX23885_IR_TX_FIFO_SERVICE_REQ 0
CYC2NS_SCALE_FACTOR 10
D64_CTRL2_AE_SHIFT 16
D64_RC_AE_SHIFT 16
D64_RC_RO_SHIFT 1
D64_XC_AE_SHIFT 16
DA7213_DMIC_CLK_RATE_SHIFT 2
DA7213_DMIC_DATA_SEL_SHIFT 0
DA7213_DMIC_SAMPLEPHASE_SHIFT 1
DA7213_MICBIAS1_LEVEL_SHIFT 0
```

```
DA7213_MICBIAS2_LEVEL_SHIFT 4
DA732X_HP_DAC_COMPO_SHIFT 3
DA9052_GPIO_EVEN_SHIFT 3
DA9052_GPIO_NIBBLE_SHIFT 4
DA9052_GPIO_ODD_SHIFT 7
DA9052_NIBBLE_SHIFT 4
DA9055_E_GPI_SHIFT 1
DA9055_RTC_MODE_SD_SHIFT 1
DA9055_TWDSCALE_SHIFT 4
DA9055_V_GPI_SHIFT 5
DAC960_BlockSizeBits 9
DAC960_MaxPartitionsBits 3
DAC_CTRL_ONDACL 4
DAC_CTRL_ONDACR 5
DAC_CTRL_ONLNOL 2
DAC_CTRL_ONLNOR 3
DAC_FORCE_DATA_SHIFT 8
DACK_POLARITY 2
DAC_PRECH_ONMSTR 0
DATA_IN_TOKEN_INTERRUPT 0
DATA_IN_TOKEN_INTERRUPT_ENABLE 0
DATA_OUT_PING_TOKEN_INTERRUPT 1
DATA_OUT_TOKEN_INTERRUPT 1
DATA_OUT_TOKEN_INTERRUPT_ENABLE 1
DATA_PACKET_RECEIVED_INTERRUPT 3
DATA_PACKET_RECEIVED_INTERRUPT_ENABLE 3
DATA_PACKET_TRANSMITTED_INTERRUPT 2
DATA_PACKET_TRANSMITTED_INTERRUPT_ENABLE 2
DATA_WIDTH 0
DB_DEF_PDU_CQPROC_SHIFT 16
DB_DEF_PDU_EVENT_SHIFT 15
DB_DEF_PDU_NUM_POSTED_SHIFT 24
DB_DEF_PDU_REARM_SHIFT 14
DB_DEF_PDU_WRB_INDEX_SHIFT 16
DB_MCCQ_NUM_POSTED_SHIFT 16
DB_RESET 26
DB_VECTOR 1
DCB_OUTPUT_ANALOG 0
DCB_OUTPUT_DP 6
DCB_OUTPUT_LVDS 3
DCB_OUTPUT_TMDS 2
DCB_OUTPUT_TV 1
DCBX_CEE_VERSION_SHIFT 12
DC_GCFG_DFHPEL_POS 12
DC_GCFG_DFHPSL_POS 8
DC_GENERAL_CFG_DFHPEL_SHIFT 12
DC_GENERAL_CFG_DFHPSL_SHIFT 8
DDS_ENT_AMP_LSB 14
DDS_ENT_MAIN_LSB 9
DDS_ENT_POST_LSB 5
DDS_ENT_PRE_LSB 0
DEBUG_SHIFT 0
DE_HAVE_BARRIER_NUMBER 0
DELAYED_UPDATE_BEACON 0
DEMAND_MODE 12
DESTROY_IN_PROGRESS 4
DEST_SHIFT 24
DEVCTRL_RTC_PWDN_SHIFT 6
DEV_ENTRY_MODE_SHIFT 9
DEVICE_CTRL_MAXRRS_MIN 2
DEVICE_REMOTE_WAKEUP_ENABLE 1
DEVICE_SET_CLEAR_DEVICE_REMOTE_WAKEUP 11
DEVICE_STATE_ENABLED_RADIO 4
DEVICE_STATE_INITIALIZED 2
DEVICE_STATE_PRESENT 0
DEVICE_STATE_REGISTERED_HW 1
```



```
DEVICE_STATE_SCANNING 5
DEVICE_STATE_STARTED 3
devnum 1
DF_BOOT_TGT 1
DFL_BLOCK_LOCKS 0
DFL_DLM_RECOVERY 6
DFL_FIRST_MOUNT 2
DFL_FIRST_MOUNT_DONE 3
DFL_MOUNT_DONE 4
DFL_NO_DLM_OPS 1
DFL_UNMOUNT 5
DF_RELOGIN 0
DIRECTION_OF_TRANSFER 3
DIRECT_LOOKUP_SHIFT 5
DISCARD_MY_DATA 21
DISCE_REVALIDATE_DOMAIN 1
DISCONNECTED 0
DISCONNECTED 8
DISCONNECT_SENT 12
DISCONN_UPCALL 16
DLC_RTR_SHIFT 6
DLM_PROC_FLAGS_CLOSING 1
DLM_PROC_FLAGS_COMPAT 2
DMA_ABORT 1
DMA_BUFFER_VALID 7
DMA_BURST_SHIFT 24
DMA_BUS_MODE_PBL_SHIFT 8
DMA_BUS_MODE_RPBL_SHIFT 17
DMA_CFG_DESC_TX_0_QID_POS 16
DMA_CHANNEL_INTERRUPT_SELECT 17
DMA_CONTROL_DACK 4
DMA_CTL_DACK 6
DMA_CTRL_DMAR_BURST_LEN_SHIFT 4
DMA_CTRL_DMAR_DLY_CNT_SHIFT 11
DMA_CTRL_DMAW_BURST_LEN_SHIFT 7
DMA_CTRL_DMAW_DLY_CNT_SHIFT 16
DMA_DIRECTION 30
DMA_DONE_INTERRUPT 6
DMA_DONE_INTERRUPT_ENABLE 29
DMA_DONE_INTERRUPT_ENABLE 6
DMAE_CMD_E1HVN_SHIFT 15
DMAE_COMMAND_C_DST_SHIFT 3
DMAE_COMMAND_C_FUNC_SHIFT 19
DMAE_COMMAND_DST_SHIFT 1
DMAE_COMMAND_DST_VFID_SHIFT 8
DMAE_COMMAND_DST_VFPF_SHIFT 14
DMAE_COMMAND_DST_VN_SHIFT 17
DMAE_COMMAND_ERR_POLICY_SHIFT 20
DMAE_COMMAND_SRC_SHIFT 0
DMAE_COMMAND_SRC_VFID_SHIFT 0
DMAE_COMMAND_SRC_VFPF_SHIFT 6
DMA_ENABLE 1
DMA_ENDPOINT_SELECT 0
DMA_EOT_ENABLE 14
DMA_FIFO_VALIDATE 2
DMA_MEMORY_WRITE_AND_INVALIDATE_ENABLE 10
DMA_MODE 0
DMA_READ_LINE_ENABLE 8
DMA_READ_MULTIPLE_ENABLE 9
DMA_REQUEST 6
DMA_REQUEST_ENABLE 5
DMAR_IQ_SHIFT 4
DMA_RWCTRL_PCI_READ_CMD_SHIFT 24
DMA_RWCTRL_PCI_WRITE_CMD_SHIFT 28
DMA_RWCTRL_READ_WATER_SHIFT 16
DMA_RWCTRL_WRITE_WATER_SHIFT 19
```

```
DMARX_CTRL_ROSHIFT 1
DMA_SCATTER_GATHER_DONE_INTERRUPT 25
DMA_START 0
DMA_TIMEOUT_ENABLE 5
DMA_TRANSACTION_DONE_INTERRUPT 24
DM_BUFIO_HASH_BITS 20
DM_CRYPT_KEY_VALID 1
DM_CRYPT_SUSPENDED 0
DMF_BLOCK_IO_FOR_SUSPEND 0
DMF_DELETING 4
DMF_FREEING 3
DMF_FROZEN 2
DMF_MERGE_IS_OPTIONAL 6
DMF_NOFLUSH_SUSPENDING 5
DMF_SUSPENDED 1
DM_KCOPYD_IGNORE_ERROR 1
DOC_ADDR_BLOCK_SHIFT 6
DONE_INTERRUPT_ENABLE 10
DOORBELL_FROM_CARD_RX 4
DPC_AEN 9
DPC_GET_DHCP_IP_ADDR 15
DPC_HA_NEED_QUIESCENT 22
DPC_HA_UNRECOVERABLE 21
DPC_LINK_CHANGED 18
DPC_POST_IDC_ACK 23
DPC_RELOGIN_DEVICE 3
DPC_RESET_ACTIVE 20
DPC_RESET_HA 1
DPC_RESET_HA_FW_CONTEXT 4
DPC_RESET_HA_INTR 5
DPC_RETRY_RESET_HA 2
DPLL_FPA01_P1_POST_DIV_SHIFT 16
DPLL_FPA01_P1_POST_DIV_SHIFT_PINEVIEW 15
DPLL_FPA1_P1_POST_DIV_SHIFT 0
DPLL_MD_UDI_MULTIPLIER_SHIFT 8
DP_TRAIN_PRE_EMPHASIS_SHIFT 3
DP_TRAIN_VOLTAGE_SWING_SHIFT 0
DQ_ACTIVE_B 5
DQ_FAKE_B 3
DQF_INFO_DIRTY_B 31
DQ_READ_B 4
DREQ_POLARITY 1
DRV_FLAGS_DCB_CONFIGURED 0
DRV_FLAGS_DCB_MFW_CONFIGURED 2
DS1621_REG_CONFIG_RESOL_SHIFT 2
DSP_DBGCNTL_EXEC_LOBIT 0
DSP_DBGCNTL_SS_LOBIT 4
DSPDMAC_DMACFG_AINCR_LOBIT 18
DSPDMAC_DMACFG_DBADR_LOBIT 0
DSPDMAC_XFRCNT_BCNT_LOBIT 16
DSPDMAC_XFRCNT_CCNT_LOBIT 0
DSP_FIFO_SR_WM_SHIFT 23
DSPFW_CURSOR_SR_SHIFT 24
DSPFW_HPLL_CURSOR_SHIFT 16
DSPFW_SR_SHIFT 23
DSP_PLANE_A_FIFO_WM1_SHIFT 16
DSP_PLANE_B_FIFO_WM1_SHIFT 24
DSP_PLANE_C_FIFO_WM_SHIFT 0
DTE_GCR3_SHIFT_A 58
DTE_GCR3_SHIFT_B 16
DTE_GCR3_SHIFT_C 43
DTIM_EXPIRED 4
DTIM_PERIOD_AVAIL 6
E1000_ADVTXD_L4LEN_SHIFT 8
E1000_ADVTXD_MACLEN_SHIFT 9
E1000_ADVTXD_MSS_SHIFT 16
```



```
E1000_ADVTXD_PAYLEN_SHIFT 14
E1000_COLD_SHIFT 12
E1000_CT_SHIFT 4
E1000_DMACR_DMACTHR_SHIFT 16
__E1000_DOWN 2
__E1000_DOWN 3
E1000_FCRTC_RTH_COAL_SHIFT 4
E1000_FWSM_MODE_SHIFT 1
E1000_GEN_CTL_ADDRESS_SHIFT 8
E1000_I2CCMD_PHY_ADDR_SHIFT 24
E1000_I2CCMD_REG_ADDR_SHIFT 16
E1000_KMRNCTRLSTA_OFFSET_SHIFT 16
E1000_LEDCTL_LED0_MODE_SHIFT 0
E1000_LTRV_NOSNOOP_SHIFT 16
E1000_LTRV_REQ_SHIFT 15
E1000_MDIC_PHY_SHIFT 21
E1000_MDIC_REG_SHIFT 16
E1000_NVM_RW_ADDR_SHIFT 2
E1000_NVM_RW_REG_DATA 16
E1000_PBA_BYTES_SHIFT 10
E1000_PSRCTL_BSIZE2_SHIFT 6
E1000_PSRCTL_BSIZE3_SHIFT 14
E1000_RCTL_MO_SHIFT 12
E1000_RTTBCNRC_RF_INT_SHIFT 14
E1000_SRRCTL_BSIZEHDRSIZE_SHIFT 2
__E1000_TESTING 0
E1000_TIMINCA_INCPERIOD_SHIFT 24
E1000_TIPG_IPGR1_SHIFT 10
E1000_TIPG_IPGR2_SHIFT 20
E1000_TX_FLAGS_VLAN_SHIFT 16
E1000_TX_HEAD_ADDR_SHIFT 7
E1000_VT_CTL_DEFAULT_POOL_SHIFT 7
E1000_VT_MSGINFO_SHIFT 16
EBLOCK_IDX_SHIFT 5
EC_FLAGS_BLOCKED 3
EC_FLAGS_GPE_STORM 1
EC_FLAGS_HANDLERS_INSTALLED 2
EC_FLAGS_QUERY_PENDING 0
EDMA_REQ_Q_PTR_SHIFT 5
EDMA_RSP_Q_PTR_SHIFT 3
EEPROM_CTRL_ADDR_SHIFT 16
__EE_WAS_ERROR 3
EFI_64BIT 5
EFI_BOOT 0
EFI_CONFIG_TABLES 2
EFI_MEMMAP 4
EFI_PAGE_SHIFT 12
EFI_RUNTIME_SERVICES 3
EFI_SYSTEM_TABLES 1
EFS_DIRBSIZE_BITS 9
EFX_FILTER_INDEX_WIDTH 13
EhnMIIdataShift 16
EhnMIIpmdShift 6
EhnMIIregShift 11
EM28XX_SNAPSHOT_KEY 212
EMAC_MDIO_MODE_CLOCK_CNT_BITSHIFT 16
EMPTYWAIT 3
Enabled 0
END_OF_CHAIN 28
ENDPOINT_0_INTERRUPT 0
ENDPOINT_0_INTERRUPT_ENABLE 0
ENDPOINT1 0
ENDPOINT_BYTE_COUNT 16
ENDPOINT_DIRECTION 4
ENDPOINT_ENABLE 10
ENDPOINT_ENABLE 7
```

```
ENDPOINT_HALT 0
ENDPOINT_NUMBER 0
ENDPOINT_TOGGLE 1
ENET_SERDES_CTRL_EMPH_0_SHIFT 4
ENET_SERDES_CTRL_EMPH_1_SHIFT 7
ENET_SERDES_CTRL_EMPH_2_SHIFT 10
ENET_SERDES_CTRL_EMPH_3_SHIFT 13
ENET_SERDES_CTRL_LADJ_0_SHIFT 16
ENET_SERDES_CTRL_LADJ_1_SHIFT 19
ENET_SERDES_CTRL_LADJ_2_SHIFT 22
ENET_SERDES_CTRL_LADJ_3_SHIFT 25
ENET_SERDES_TEST_MD_0_SHIFT 0
ENET_SERDES_TEST_MD_1_SHIFT 2
ENET_SERDES_TEST_MD_2_SHIFT 4
ENET_SERDES_TEST_MD_3_SHIFT 6
ENTRY_BITPOS_DESCRIPTOR 10
ENTRY_BITPOS_QWORDS 0
ENTRY_DATA_IO_FAILED 3
ENTRY_DATA_PENDING 2
ENTRY_DATA_STATUS_PENDING 4
ENTRY_DATA_STATUS_SET 5
ENTRY_OWNER_DEVICE_DATA 1
EOT_POLARITY 3
EP_DISC_ABORT 18
EP_DISC_CLOSE 17
EP_FLAG_RUNNING 1
EP_FLAG_STOPPING 2
EPLD_DMA_ENABLE 7
EP_QUEUE_HEAD_MAX_PKT_LEN_POS 16
EP_QUEUE_HEAD_MULT_POS 30
ESD_USB2_TSEG1_SHIFT 16
ESD_USB2_TSEG2_SHIFT 20
ESPC_PIO_STAT_ADDR_SHIFT 8
ESR_GLUE_CTRL0_BLTIME_SHIFT 24
ESR_GLUE_CTRL0_SRATE_SHIFT 8
ESR_GLUE_CTRL0_THCNT_SHIFT 0
ESR_RXTX_CTRL_VMUXLO_SHIFT 22
ESTAB_UPCALL 7
ETH_CLASSIFY_CMD_HEADER_OPCODE_SHIFT 2
ETH_TX_BD_FLAGS_VLAN_MODE_SHIFT 2
ETH_TX_PARSE_2ND_BD_IP_HDR_LEN_OUTER_W_SHIFT 8
ETH_TX_PARSE_2ND_BD_IP_HDR_TYPE_OUTER_SHIFT 4
ETH_TX_PARSE_2ND_BD_LLC_SNAP_EN_SHIFT 5
ETH_TX_PARSE_BD_E1X_ETH_ADDR_TYPE_SHIFT 4
ETH_TX_PARSE_BD_E2_ETH_ADDR_TYPE_SHIFT 30
ETH_TX_PARSE_BD_E2_L4_HDR_START_OFFSET_W_SHIFT 0
ETH_TX_PARSE_BD_E2_LSO_MSS_SHIFT 16
ETH_TX_PARSE_BD_E2_TCP_HDR_LENGTH_DW_SHIFT 11
ETH_TX_START_BD_HDR_NBDS_SHIFT 0
ET_MAC_CFG2_PREAMBLE_SHIFT 12
ET_MAC_STATION_ADDR1_OC4_SHIFT 8
ET_MAC_STATION_ADDR1_OC5_SHIFT 16
ET_MAC_STATION_ADDR1_OC6_SHIFT 24
ET_MAC_STATION_ADDR2_OC1_SHIFT 16
ET_MAC_STATION_ADDR2_OC2_SHIFT 24
ET_RX_PFCTRL_MIN_PKT_SZ_SHIFT 16
ET_RX_UNI_PF_ADDR1_1_SHIFT 8
ET_RX_UNI_PF_ADDR1_3_SHIFT 24
ET_RX_UNI_PF_ADDR1_4_SHIFT 16
ET_RX_UNI_PF_ADDR1_5_SHIFT 8
ET_RX_UNI_PF_ADDR2_1_SHIFT 24
ET_RX_UNI_PF_ADDR2_2_SHIFT 16
ET_RX_UNI_PF_ADDR2_3_SHIFT 24
ET_RX_UNI_PF_ADDR2_4_SHIFT 16
ET_RX_UNI_PF_ADDR2_5_SHIFT 8
ET_RX_WOL_HI_SA1_SHIFT 8
```



```
ET_RX_WOL_LO_SA3_SHIFT 24
ET_RX_WOL_LO_SA4_SHIFT 16
ET_RX_WOL_LO_SA5_SHIFT 8
ET_TXDMA_CACHE_SHIFT 4
EV_ABS 3
EVENT_DEV_ASLEEP 6
EVENT_DEV_OPEN 7
EVENT_NO_RUNTIME_PM 9
EVENT_RX_KILL 10
EVENT_RX_PAUSED 5
EVENT_STS_SPLIT 3
EV_KEY 1
EV_LED 17
EV_MSC 4
EV_REL 2
EV_REP 20
EV_SW 5
EV_SYN 0
EXHCH_HSIZE_SHIFT 0
EXHCH_VSIZE_SHIFT 2
EXT4_GROUP_INFO_NEED_INIT_BIT 0
EXTENT_BUFFER_CORRUPT 3
EXTENT_BUFFER_DUMMY 9
EXTENT_BUFFER_IOERR 8
EXTENT_BUFFER_READAHEAD 4
EXTENT_BUFFER_STALE 6
EXTENT_BUFFER_UPTODATE 0
EXTENT_BUFFER_WRITEBACK 7
EXTENT_FLAG_COMPRESSED 1
EXTENT_FLAG_FILLING 5
EXTENT_FLAG_LOGGING 4
EXTENT_FLAG_PINNED 0
EXTENT_FLAG_PREALLOC 3
EXTENT_FLAG_VACANCY 2
EXTIN_AC97_L 0
EXTIN_AC97_R 1
EXTIN_COAX_SPDIF_L 10
EXTIN_COAX_SPDIF_R 11
EXTIN_LINE1_L 8
EXTIN_LINE1_R 9
EXTIN_LINE2_L 12
EXTIN_LINE2_R 13
EXTIN_SPDIF_CD_L 2
EXTIN_SPDIF_CD_R 3
EXTIN_TOSLINK_L 6
EXTIN_TOSLINK_R 7
EXTIN_ZOOM_L 4
EXTIN_ZOOM_R 5
EXTOUT_AC97_CENTER 4
EXTOUT_AC97_L 0
EXTOUT_AC97_LFE 5
EXTOUT_AC97_R 1
EXTOUT_AC97_REAR_L 13
EXTOUT_AC97_REAR_R 14
EXTOUT_HEADPHONE_L 6
EXTOUT_HEADPHONE_R 7
EXTOUT_MIC_CAP 12
EXTOUT_REAR_L 8
EXTOUT_REAR_R 9
EXTOUT_TOSLINK_L 2
EXTOUT_TOSLINK_R 3
EXTRA_FLAGS 16
FAM10H_MMIO_CONF_BASE_SHIFT 20
FAM10H_MMIO_CONF_BUSRANGE_SHIFT 2
FAST_SLOW_TERMINATE_MODE_SELECT 15
FaultRecorded 7
```

```
Faulty 0
FCOE_CQE_TOGGLE_BIT_SHIFT 15
FCOE_CTX_RESET_NEEDED 18
FCOE_IOS_PER_CONNECTION_SHIFT 0
FCOE_KWQE_CONN_ENABLE_DISABLE_PRIORITY_SHIFT 13
FCOE_KWQE_CONN_ENABLE_DISABLE_VLAN_ID_SHIFT 0
FCOE_KWQE_CONN_OFFLOAD3_B_CONF_REQ_SHIFT 3
FCOE_KWQE_CONN_OFFLOAD3_B_CONT_INCR_SEQ_CNT_SHIFT 2
FCOE_KWQE_CONN_OFFLOAD3_B_E_D_TOV_RES_SHIFT 1
FCOE_KWQE_CONN_OFFLOAD3_B_REC_VALID_SHIFT 4
FCOE_KWQE_CONN_OFFLOAD3_B_VLAN_FLAG_SHIFT 7
FCOE_KWQE_CONN_OFFLOAD3_PRIORITY_SHIFT 13
FCOE_KWQE_CONN_OFFLOAD3_VLAN_ID_SHIFT 0
FCOE_KWQE_HEADER_LAYER_CODE_SHIFT 4
FCOE_KWQE_INIT1_LOG_PAGE_SIZE_SHIFT 0
FCOE_LOGINS_PER_PORT_SHIFT 16
FCOE_NPIV_WWN_PER_PORT_SHIFT 16
FCOE_NUMBER_OF_EXCHANGES_SHIFT 0
FCOE_OUTSTANDING_COMMANDS_SHIFT 16
FCOE_SQE_TASK_ID_SHIFT 0
FCOE_SQE_TOGGLE_BIT_SHIFT 15
FCOE_TARGETS_SUPPORTED_SHIFT 0
FCOE_TCE_RX_WR_TX_RD_CONST_CID_SHIFT 0
FCOE_TCE_RX_WR_TX_RD_VAR_EXP_FIRST_FRAME_SHIFT 12
FCOE_TCE_TX_WR_RX_RD_CONST_CACHED_SGE_SHIFT 5
FCOE_TCE_TX_WR_RX_RD_CONST_CLASS_TYPE_SHIFT 4
FCOE_TCE_TX_WR_RX_RD_CONST_DEV_TYPE_SHIFT 3
FCOE_TCE_TX_WR_RX_RD_CONST_TASK_TYPE_SHIFT 0
FCOE_TCE_TX_WR_RX_RD_CONST_TX_STATE_SHIFT 1
FCPORT_UPDATE_NEEDED 13
FCRAM_REF_TMR_MAX_SHIFT 16
FCRAM_REF_TMR_MIN_SHIFT 0
FD_DISK_CHANGED_BIT 4
FD_DISK_NEWCHANGE_BIT 2
FD_DISK_WRITABLE_BIT 5
FD_NEED_TWADDLE_BIT 0
FD_VERIFY_BIT 1
FF_AUTOCENTER 97
FF_CONSTANT 82
FF_CORE_SHOULD_PLAY 4
FF_CORE_UPDATE 5
FF_DAMPER 85
FF_FRICTION 84
FF_GAIN 96
FF_INERTIA 86
FFLP_CFG_1_CAMLAT_SHIFT 16
FFLP_CFG_1_CAMRATIO_SHIFT 12
FFLP_CFG_1_FCRAMRATIO_SHIFT 8
FF_MOD1_IS_USED 0
FF_MOD2_IS_USED 1
FF_PERIODIC 81
FF_RAMP 87
FF_RUMBLE 80
FF_SAW_DOWN 92
FF_SAW_UP 91
FF_SINE 90
FF_SPRING 83
FF_SQUARE 88
FF_TRIANGLE 89
FH_MEM_TFDIB_REG1_ADDR_BITSHIFT 28
FIELD_SIZE 5
FIFO_EMPTY 10
FIFO_FLUSH 9
FIFO_FULL 11
FIFO_OVERFLOW 13
FIFOSIZE_DEPTH_SHIFT 16
```



```
FIFOSIZE_STARTADDR_SHIFT 0
FIFO_SOFT_RESET 4
FIFO_UNDERFLOW 12
FIP_DT_MAC 2
FIRST_BOOT 7
FirstUse 10
FLAG_802_11 7
FLAG_ACTIVE_LOW 6
FLAG_ADHOC 3
FLAG_COMMIT 13
FLAG_EFS_ENABLE 5
FLAG_ENABLED 2
FLAG_EXITING 0
FLAG_EXPORT 2
FLAG_EXT_CTRL 4
FLAG_FLASHING 15
FLAG_FLUSHABLE 3
FLAG_FORCE_ACTIVE 1
FLAG_FORCE_RELIABLE 2
FLAG_IRQ_ENABLE 1
FLAG_IS_OUT 1
FLAG_MIC_CAPABLE 4
FLAG_MPI 11
FLAG_OPEN_DRAIN 7
FLAG_OPEN_SOURCE 8
FLAG_PENDING_XMIT11 10
FLAG_PENDING_XMIT 9
FLAG_RADIO_DOWN 1
FLAG_RADIO_OFF 0
FLAG_REGISTERED 12
FLAG_RESET 14
FLAG_ROLE_SWITCH 0
FLAG_SYSFS 3
FLAG_UPDATE_MULTI 5
FLAG_UPDATE_UNI 6
FLAG_VBUS_CHANGED 0
FLAG_WPA_CAPABLE 16
FLASH_SECTOR_ADDR_SHIFT 12
FLG_ACTIVE 6
FLG_HDLC 13
FLG_L1_DBUSY 5
FLG_L1_PULL_REQ 6
FLG_TRANSPARENT 12
FLIDX_ABORTING 2
FLIDX_DISCONNECTING 3
FLIDX_SG_ACTIVE 1
FLIDX_TIMED_OUT 5
FLIDX_URB_ACTIVE 0
FLOW_KEY_L4_0_SHIFT 2
FLOW_KEY_L4_1_SHIFT 0
FLT_SHIFT 0
FLW_PRT_SEL_BASE_SHIFT 0
FLW_PRT_SEL_MASK_SHIFT 8
FM801_AC97_ADDR_SHIFT 10
FM801_RATE_SHIFT 8
FM_AF_SWITCH_INPROGRESS 5
FM_CORE_READY 3
FM_CORE_TX_XMITING 6
FM_FW_DW_INPROGRESS 2
FM_INTTASK_RUNNING 0
FM_INTTASK_SCHEDULE_PENDING 1
FNIC_INTX_ERR 1
FNIC_INTX_NOTIFY 2
FNIC_INTX_WQ_RQ_COPYWQ 0
FORCE_DETACH 14
FORCE_FULL_SPEED_MODE 30
```

```
FORCE_HIGH_SPEED 3
FP_PT1_VSIZE_SHIFT 16
FP_SHIFT 12
FRAC_BITS 14
FRAC_BITS 8
FREQ_RATIO_OFFSET 19
FS155_VCI_BITS 6
FS155_VPI_BITS 4
FSCACHE_COOKIE_LOOKING_UP 0
FSCACHE_COOKIE_NO_DATA_YET 1
FSCACHE_COOKIE_RELINQUISHED 4
FSCACHE_COOKIE_RETIRED 5
FSCACHE_COOKIE_UNAVAILABLE 2
FSCACHE_OBJECT_EV_CLEARED 4
FSCACHE_OBJECT_IS_AVAILABLE 5
FSCACHE_OBJECT_IS_LIVE 3
FSCACHE_OP_EXCLUSIVE 5
FSCACHE_OP_UNUSE_COOKIE 7
FSCACHE_OP_WAITING 4
FSHIFT 11
FTRACE_EVENT_FL_ENABLED_BIT 0
FTRACE_EVENT_FL_RECORDED_CMD_BIT 1
FTRACE_EVENT_FL_SOFT_DISABLED_BIT 3
FTRACE_EVENT_FL_SOFT_MODE_BIT 2
FULLNESS_BITS 4
FULL_SPEED 6
FUNC_MF_CFG_MAX_BW_SHIFT 24
FUSE_I_ADVISE_RDPLUS 0
FW_STATUS_ABORT 2
FW_STATUS_DONE 1
FW_STATUS_LOADING 0
FX00_RESET_RECOVERY 25
FX00_TARGET_SCAN 26
GAIN_ALPHA 5
GBF_FULL 1
GDFIFOCFG_EPINFOBASE_SHIFT 16
GEN6_BLITTER_LOCK_SHIFT 16
GEN6_MBC_SNPCR_SHIFT 21
GEN6_PCODE_FREQ_IA_RATIO_SHIFT 8
GEN6_PCODE_FREQ_RING_RATIO_SHIFT 16
GENERATE_RESUME 3
GENERATE_RESUME 5
GET_DEVICE_STATUS 0
GET_INTERFACE_STATUS 1
GI2CCTL_I2CDEVADDR_SHIFT 26
GIF_ALLOC_FAILED 2
GIF_INVALID 0
GIF_QD_LOCKED 1
GIF_SW_PAGED 3
GLF_BLOCKING 15
GLF_DEMOTE 3
GLF_DEMOTE_IN_PROGRESS 5
GLF_DIRTY 6
GLF_FROZEN 11
GLF_INITIAL 10
GLF_INVALIDATE_IN_PROGRESS 8
GLF_LFLUSH 7
GLF_LRU 13
GLF_QUEUED 12
GLF_REPLY_PENDING 9
GLOBAL_BEEP_ENABLE_SHIFT 15
GMAC_FLOW_CTRL_PT_SHIFT 16
GMBUS_SLAVE_INDEX_SHIFT 8
GMCR_GAPBB_SHIFT 14
GMCR_GAPR1_SHIFT 7
GMCR_GAPR2_SHIFT 0
```



```
GPIO_LLI_BIT 5
GPIO_MIC_RELAY 4
GPIO_OVRUN_BIT 4
GPIO_RXUVL_BIT 3
GRC_MISC_CFG_PRESCALAR_SHIFT 1
GRCR_HASHSIZE_SHIFT 17
GREG_SWRST_CACHE_SHIFT 16
GRSTCTL_TXFNUM_SHIFT 6
GS40G_PAGE_SHIFT 16
H5_RX_ESC 0
H5_TX_ACK_REQ 1
HAS_ALARM 1
HASH_FN_SHIFT 13
HAS_NVRAM 0
HCCHAR_DEVADDR_SHIFT 22
HCCHAR_EPNUM_SHIFT 11
HCCHAR_EPTYPE_SHIFT 18
HCCHAR_MPS_SHIFT 0
HCCHAR_MULTICNT_SHIFT 20
HCD_FLAG_DEAD 6
HCD_FLAG_HW_ACCESSIBLE 0
HCD_FLAG_POLL_PENDING 3
HCD_FLAG_POLL_RH 2
HCD_FLAG_RH_RUNNING 5
HCD_FLAG_WAKEUP_PENDING 4
HCI_AUTH 5
HCI_AUTO_OFF 1
HCI_CONNECTABLE 12
HCI_CONN_ENCRYPT_PEND 2
HCI_CONN_POWER_SAVE 9
HCI_CONN_REAUTH_PEND 1
HCI_CONN_REMOTE_OOB 10
HCI_CONN_SCO_SETUP_PEND 5
HCI_CONN_SSP_ENABLED 8
HCI_DEBUG_KEYS 5
HCI_ENCRYPT 6
HCI_FAST_CONNECTABLE 16
HCI_HS_ENABLED 9
HCI_INIT 1
HCI_INQUIRY 7
HCI_ISCAN 4
HCI_LE_ENABLED 10
HCI_LE_PERIPHERAL 11
HCI_LE_SCAN 7
HC_INDEX_DATA_HC_ENABLED_SHIFT 1
HC_INDEX_DATA_SM_ID_SHIFT 0
HCI_PAIRABLE 3
HCI_PERIODIC_INQ 15
HCI_PSCAN 3
HCI_QUIRK_FIXUP_BUFFER_SIZE 2
HCI_QUIRK_RAW_DEVICE 1
HCI_QUIRK_RESET_ON_CLOSE 0
HCI_RAW 8
HCI_RESET 9
HCI_RUNNING 2
HCI_SETUP 0
HCI_SSP_ENABLED 8
HCI_UART_INIT_PENDING 3
HCI_UART_REGISTERED 1
HCI_UART_SENDING 1
HCI_UART_TX_WAKEUP 2
HCI_UNREGISTER 6
HCI_UP 0
HCSPLT_HUBADDR_SHIFT 7
HCSPLT_PRTADDR_SHIFT 0
HCSPLT_XACTPOS_SHIFT 14
```

```
HDPVR_RCA_BACK 0
HDPVR_RCA_FRONT 1
HDPVR_SPDIF 2
HERMES_MIC_KEY_ID_SHIFT 11
HFIR_FRINT_SHIFT 0
HFS_BNODE_DELETED 2
HFS_BNODE_DELETED 4
HFS_BNODE_ERROR 0
HFS_BNODE_ERROR 1
HFS_BNODE_NEW 1
HFS_BNODE_NEW 2
HFS_FLG_ALT_MDB_DIRTY 2
HFS_FLG_MDB_DIRTY 1
HFSPLUS_I_CAT_DIRTY 1
HFSPLUS_I_EXT_DIRTY 2
HFSPLUS_I_RSRC 0
HFSPLUS_SB_CASEFOLD 4
HFSPLUS_SB_FORCE 2
HFSPLUS_SB_HFSX 3
HFSPLUS_SB_NOBARRIER 5
HFSPLUS_SB_NODECOMPOSE 1
HFSPLUS_SB_WRITEBACKUP 0
HFSPLUS_SECTOR_SHIFT 9
HFS_SECTOR_SIZE_BITS 9
HID_CLEAR_HALT 6
HID_CTRL_RUNNING 1
HID_DISCONNECTED 7
HIDE_STATUS_PHASE 6
HID_KEYS_PRESSED 10
HID_NO_BANDWIDTH 11
HID_OUT_RUNNING 2
HIDP_BLUETOOTH_VENDOR_ID 9
HIDP_VIRTUAL_CABLE_UNPLUG 0
HIDP_WAITING_FOR_RETURN 10
HIDP_WAITING_FOR_SEND_ACK 11
HID_RESET_PENDING 4
HID_STARTED 8
HID_SUSPENDED 5
HI_EARLY_ALLOC_IRAM_BANKS_SHIFT 0
HI_EARLY_ALLOC_MAGIC_SHIFT 16
HIF_HOLDER 6
HIGH_SPEED 7
HighThresholdShift 2
HI_OPTION_FW_BRIDGE_SHIFT 4
HI_OPTION_FW_MODE_SHIFT 12
HI_OPTION_FW_SUBMODE_SHIFT 20
HI_OPTION_MAC_ADDR_METHOD_SHIFT 3
HI_OPTION_NUM_DEV_SHIFT 9
HMC5843_RANGE_GAIN_OFFSET 5
HMC5843_RATE_OFFSET 2
HORZ_PANEL_SHIFT 16
HOST_CONTROL_IF_SHIFT 4
HOST_DMA_ISOC_NBYTES_SHIFT 0
HOST_DMA_NBYTES_SHIFT 0
HOST_RAMP_DOWN_QUEUE_DEPTH 22
HOST_SLEEP_MODE_CMD_PROCESSED 9
HREF_HSIZE_SHIFT 0
HREF_HSTART_SHIFT 4
HREF_VSIZE_SHIFT 2
HREF_VSTART_SHIFT 6
HSO_NET_RUNNING 0
HT_AGG_STATE_OPERATIONAL 2
HT_AGG_STATE_STOPPING 3
HT_AGG_STATE_WANT_STOP 5
HTC_CONN_FLGS_SET_RECV_ALLOC_SHIFT 8
HTC_MAILBOX 0
```



```
HTS2_PRST_SHIFT 10
HTS_PCPL_SHIFT 21
HTS_PTL_SHIFT 1
HV_KMRN_FIFO_CTRLSTA_PREAMBLE_SHIFT 12
HV_SMB_ADDR_FREQ_LOW_SHIFT 8
HW_HT_RATES_OFFSET 16
HW_MIMO_RATES_OFFSET 24
HWTSTAMP_FILTER_ALL 1
HWTSTAMP_FILTER_NONE 0
HWTSTAMP_FILTER_PTP_V1_L4_DELAY_REQ 5
HWTSTAMP_FILTER_PTP_V1_L4_EVENT 3
HWTSTAMP_FILTER_PTP_V1_L4_SYNC 4
HWTSTAMP_FILTER_PTP_V2_DELAY_REQ 14
HWTSTAMP_FILTER_PTP_V2_EVENT 12
HWTSTAMP_FILTER_PTP_V2_L2_DELAY_REQ 11
HWTSTAMP_FILTER_PTP_V2_L2_EVENT 9
HWTSTAMP_FILTER_PTP_V2_L2_SYNC 10
HWTSTAMP_FILTER_PTP_V2_L4_DELAY_REQ 8
HWTSTAMP_FILTER_PTP_V2_L4_EVENT 6
HWTSTAMP_FILTER_PTP_V2_L4_SYNC 7
HWTSTAMP_FILTER_PTP_V2_SYNC 13
HWTSTAMP_TX_OFF 0
HWTSTAMP_TX_ON 1
HWTSTAMP_TX_ONESTEP_SYNC 2
I2C_HID_READ_PENDING 4
I2C_HID_RESET_PENDING 2
I2C_HID_STARTED 1
i2c_m_status_wdat_done_pos 1
i2c_m_status_wdat_fail_pos 2
I2F_FRAC_BITS 23
I830_FENCE_PITCH_SHIFT 4
I830_FENCE_TILING_Y_SHIFT 12
I965_FENCE_TILING_Y_SHIFT 1
IA32_SYSCALL_VECTOR 128
IBA6120_R_PKEY_DIS_SHIFT 30
IBA7220_IBCC_LINKCMD_SHIFT 19
IBA7220_IBC_RXPOL_SHIFT 7
IBA7220_LEDBLINK_OFF_SHIFT 0
IBA7220_LEDBLINK_ON_SHIFT 32
IBA7220_R_CTXTCFG_SHIFT 36
IBA7220_R_INTRAVAIL_SHIFT 17
IBA7220_R_PKEY_DIS_SHIFT 34
IBA7220_R_TAILUPD_SHIFT 35
IBA7220_TID_PA_SHIFT 11
IBA7220_TID_SZ_SHIFT 37
IBA7322_HDRHEAD_PKTINT_SHIFT 32
IBA7322_TID_PA_SHIFT 11
IBA7322_TID_SZ_SHIFT 37
IB_ACK_REQUESTED 1
IB_MGMT_METHOD_GET 1
IB_MGMT_METHOD_SEND 3
IB_MGMT_METHOD_SET 2
IBSHIFT 16
IBS_STARTED 1
IBS_STOPPING 2
IB_USER_VERBS_CMD_ALLOC_MW 14
IB_USER_VERBS_CMD_ALLOC_PD 3
IB_USER_VERBS_CMD_ATTACH_MCAST 30
IB_USER_VERBS_CMD_BIND_MW 15
IB_USER_VERBS_CMD_CLOSE_XRCD 38
IB_USER_VERBS_CMD_CREATE_AH 5
IB_USER_VERBS_CMD_CREATE_COMP_CHANNEL 17
IB_USER_VERBS_CMD_CREATE_CQ 18
IB_USER_VERBS_CMD_CREATE_QP 24
IB_USER_VERBS_CMD_CREATE_SRQ 32
IB_USER_VERBS_CMD_CREATE_XSRQ 39
```

```
IB_USER_VERBS_CMD_DEALLOC_MW 16
IB_USER_VERBS_CMD_DEALLOC_PD 4
IB_USER_VERBS_CMD_DEREG_MR 13
IB_USER_VERBS_CMD_DESTROY_AH 8
IB_USER_VERBS_CMD_DESTROY_CQ 20
IB_USER_VERBS_CMD_DESTROY_QP 27
IB_USER_VERBS_CMD_DESTROY_SRQ 35
IB_USER_VERBS_CMD_DETACH_MCAST 31
IB_USER_VERBS_CMD_GET_CONTEXT 0
IB_USER_VERBS_CMD_MODIFY_AH 6
IB_USER_VERBS_CMD_MODIFY_QP 26
IB_USER_VERBS_CMD_MODIFY_SRQ 33
IB_USER_VERBS_CMD_OPEN_QP 40
IB_USER_VERBS_CMD_OPEN_XRCD 37
IB_USER_VERBS_CMD_POLL_CQ 21
IB_USER_VERBS_CMD_POST_RECV 29
IB_USER_VERBS_CMD_POST_SEND 28
IB_USER_VERBS_CMD_POST_SRQ_RECV 36
IB_USER_VERBS_CMD_QUERY_AH 7
IB_USER_VERBS_CMD_QUERY_DEVICE 1
IB_USER_VERBS_CMD_QUERY_PORT 2
IB_USER_VERBS_CMD_QUERY_QP 25
IB_USER_VERBS_CMD_QUERY_SRQ 34
IB_USER_VERBS_CMD_REG_MR 9
IB_USER_VERBS_CMD_REQ_NOTIFY_CQ 23
IB_USER_VERBS_CMD_RESIZE_CQ 19
ICH_DI1L_SHIFT 4
ICH_DI2L_SHIFT 6
IDE_CAST_CMD_SHIFT 24
IDEV_ABORT_PATH_ACTIVE 7
IDEV_ABORT_PATH_RESUME_PENDING 8
IDEV_GONE 3
IDEV_IO_NCQERROR 5
IDEV_IO_READY 4
IDEV_START_PENDING 0
IDEV_STOP_PENDING 1
IDLE_CODE 25
ID_SHIFT 16
IDT77252_BIT_INIT 1
IEEE80211_BAR_CTRL_TID_INFO_SHIFT 12
IEEE80211_HT_AMPDU_PARM_DENSITY_SHIFT 2
IEEE80211_HT_CAP_RX_STBC_SHIFT 8
IEEE80211_HT_CAP_SM_PS_SHIFT 2
IEEE80211_HT_MCS_TX_MAX_STREAMS_SHIFT 2
IEEE80211_RADIOTAP_EXT 31
IEEE80211_RADIOTAP_MCS_STBC_SHIFT 5
IEEE80211_SEQ_SEQ_SHIFT 4
IEEE80211_TX_CTL_STBC_SHIFT 23
IEEE80211_VHT_CAP_MAX_A_MPDU_LENGTH_EXPONENT_SHIFT 23
IEEE80211_WMM_IE_STA_QOSINFO_SP_SHIFT 5
IEEE802154_FC_DAMODE_SHIFT 10
IEEE802154_FC_SAMODE_SHIFT 14
IFORCE_XMIT_AGAIN 1
IFX_SPI_DTR 4
IFX_SPI_MORE_BIT 4
IFX_SPI_RTS 5
IFX_SPI_STATE_IO_AVAILABLE 4
IFX_SPI_STATE_IO_READY 2
IFX_SPI_STATE_PRESENT 0
IFX_SPI_UPDATE 8
IGB_82576_TSYNC_SHIFT 19
__IGB_DOWN 2
IGB_RING_FLAG_RX_LB_VLAN_BSWAP 1
IGB_RING_FLAG_RX_SCTP_CSUM 0
IGB_RING_FLAG_TX_CTX_IDX 2
IGB_RING_FLAG_TX_DETECT_HANG 3
```



```
__IGB_TESTING 0
IGB_TX_FLAGS_VLAN_SHIFT 16
__IGBVF_DOWN 2
__IGBVF_TESTING 0
IGBVF_TX_FLAGS_VLAN_SHIFT 16
IGNORE_NEXT_INT 2
IGP_PAGE_SHIFT 5
IGU_ACK_REGISTER_INTERRUPT_MODE_SHIFT 9
IGU_ACK_REGISTER_STATUS_BLOCK_ID_SHIFT 0
IGU_ACK_REGISTER_STORM_ID_SHIFT 5
IGU_ACK_REGISTER_UPDATE_INDEX_SHIFT 8
IGU_CTRL_REG_ADDRESS_SHIFT 0
IGU_CTRL_REG_FID_SHIFT 12
IGU_CTRL_REG_TYPE_SHIFT 20
IGU_FID_ENCODE_IS_PF_SHIFT 6
IGU_REGULAR_BUPDATE_SHIFT 24
IGU_REGULAR_CLEANUP_TYPE_SHIFT 28
IGU_REGULAR_ENABLE_INT_SHIFT 25
IGU_REGULAR_SB_INDEX_SHIFT 0
IGU_REGULAR_SEGMENT_ACCESS_SHIFT 21
IGU_VF_CONF_PARENT_SHIFT 2
IHFLG_IFSHFT 2
IHOST_IRQ_ENABLED 2
IHOST_START_PENDING 0
IHOST_STOP_PENDING 1
IL_FIRST_OFDM_RATE 4
IMG_REQ_CHILD 1
IMG_REQ_LAYERED 2
IMG_REQ_WRITE 0
IMGSZ_V_SHIFT 16
IMM 6
INFINIPATH_HWE_HTCMEMPARITYERR_SHIFT 0
INFINIPATH_HWE_RXEMEMPARITYERR_SHIFT 44
INFINIPATH_HWE_TXEMEMPARITYERR_SHIFT 40
INFINIPATH_IBCC_CREDITSCALE_SHIFT 40
INFINIPATH_IBCC_FLOWCTRLPERIOD_SHIFT 0
INFINIPATH_IBCC_FLOWCTRLWATERMARK_SHIFT 8
INFINIPATH_IBCC_LINKINITCMD_SHIFT 16
INFINIPATH_IBCC_OVERRUNTHRESHOLD_SHIFT 36
INFINIPATH_IBCC_PHYERRTHRESHOLD_SHIFT 32
INFINIPATH_IBCS_LINKTRAININGSTATE_SHIFT 0
INFINIPATH_I_RCVAVAIL_SHIFT 12
INFINIPATH_I_RCVURG_SHIFT 0
INFINIPATH_RT_BUFSIZE_SHIFT 48
INFINIPATH_S_DISARMPIOBUF_SHIFT 16
INFINIPATH_SENDPIOAVAIL_BUSY_SHIFT 1
INFINIPATH_S_UPDTHRESH_SHIFT 24
INFINIPATH_XGXS_RX_POL_SHIFT 19
INIT_HCA_DEVICE_MANAGED_FLOW_STEERING_EN 6
INIT_IB_MTU_SHIFT 12
INIT_IB_PORT_WIDTH_SHIFT 8
INIT_IB_VL_SHIFT 4
INIT_PORT_PORT_WIDTH_SHIFT 8
INIT_PORT_VL_SHIFT 4
INPUT_PROP_SEMI_MT 3
In_sync 1
INTA_ASSERTED 12
INTEL_ANALOG_CLONE_BIT 9
INTEL_OUTPUT_ANALOG 1
INTEL_OUTPUT_DISPLAYPORT 9
INTEL_OUTPUT_EDP 10
INTEL_OUTPUT_HDMI 6
INTEL_OUTPUT_LVDS 4
INTEL_OUTPUT_MIPI2 8
INTEL_OUTPUT_MIPI 7
INTEL_OUTPUT_SDVO 3
```

```
INTEL_OUTPUT_TVOUT 5
INTEL_PMC_IDX_FIXED 32
INTEL_SDVO_LVDS_CLONE_BIT 8
INTEL_SDVO_NON_TV_CLONE_BIT 6
INTEL_SDVO_TV_CLONE_BIT 7
INT_EN 1
INTERCEPT_IOIO_PROT 27
INTERCEPT_MSR_PROT 28
INTERCEPT_NMI 1
INTERCEPT_SELECTIVE_CR0 5
INTERCEPT_VMRUN 32
INTERRUPT_AFTER_TERMINAL_COUNT 2
INTERRUPT_MODE 4
INT_MASK_WE 8
INTR_COAL_COUNT_SHIFT 16
INV_MPU6050_ACCL_CONFIG_FSR_SHIFT 3
INV_MPU6050_GYRO_CONFIG_FSR_SHIFT 3
IOAT_COMPLETION_ACK 1
IOAT_COMPLETION_PENDING 0
IOAT_KOBJ_INIT_FAIL 3
IOAT_MMIO_BAR 0
IOAT_RESET_PENDING 2
IOAT_RUN 5
_IOC_NR 101
_IOC_NR 102
_IOC_NR 2
_IOC_NR 21
_IOC_NR 25
_IOC_NR 27
_IOC_NR 28
_IOC_NR 36
_IOC_NR 37
_IOC_NR 4
_IOC_NR 5
_IOC_NR 58
_IOC_NR 59
_IOC_NR 60
_IOC_NR 64
_IOC_NR 67
_IOC_NR 68
_IOC_NR 71
_IOC_NR 72
_IOC_NR 73
_IOC_NR 79
_IOC_NR 80
IOCR_INPUTS_OFFSET 8
IOCR_OUTPUTS_OFFSET 0
IO_DONE_BIT 0
IO_ERROR_SHIFT 20
IOMMU_CAP_EFR 27
IOMMU_CAP_IOTLB 24
IOMMU_CAP_NPCACHE 26
IOPM_ALLOC_ORDER 2
IOPOLL_F_DISABLE 1
IOPOLL_F_SCHED 0
IOSF_BAR_SHIFT 1
IOSF_BYTE_ENABLES_SHIFT 4
IOSF_DEVFN_SHIFT 24
IOSF_OPCODE_SHIFT 16
IOSF_PORT_SHIFT 8
IO_TLB_SHIFT 11
IO_WAKEUP_ENABLE 1
IPATH_AETH_CREDIT_SHIFT 24
IPATH_GPIO_LLI_BIT 5
IPATH_GPIO_OVRUN_BIT 4
IPATH_GPIO_PORT0_BIT 2
```



```
IPATH_GPIO_RXUVL_BIT 3
IPATH_PORT_MASTER_UNINIT 4
IPATH_PORT_WAITING_RCV 2
IPATH_PORT_WAITING_URG 5
IPATH_R_WRID_VALID 0
IPATH_SDMA_DISARMED 1
IPCT_ASSURED 4
IPCT_DESTROY 2
IPCT_HELPER 6
IPCT_LABEL 10
IPCT_MARK 7
IPCT_NATSEQADJ 8
IPCT_NEW 0
IPCT_PROTOINFO 5
IPCT_RELATED 1
IPCT_REPLY 3
IPCT_SECMARK 9
IPEXP_DESTROY 1
IPEXP_NEW 0
IPMI_FLAGS_HANDLER_INSTALL 0
IPOIB_FLAG_ADMIN_CM 9
IPOIB_FLAG_ADMIN_UP 2
IPOIB_FLAG_INITIALIZED 1
IPOIB_FLAG_OPER_UP 0
IPOIB_FLAG_SUBINTERFACE 5
IPOIB_FLAG_UMCAST 10
IPOIB_MCAST_FLAG_BUSY 2
IPOIB_MCAST_FLAG_FOUND 0
IPOIB_MCAST_FLAG_SENDONLY 1
IPOIB_NEIGH_TBL_FLUSH 12
IPOIB_PKEY_ASSIGNED 3
IPOIB_PKEY_STOP 4
IPOIB_STOP_NEIGH_GC 11
IPOIB_STOP_REAPER 7
IPORT_RESET_PENDING 0
IPP_CFIG_IP_MAX_PKT_SHIFT 8
IPS_ASSURED_BIT 2
IPS_DYING_BIT 9
IPS_HELPER_BIT 13
IPS_SEQ_ADJUST_BIT 6
IP_VS_DEST_STATE_REMOVING 1
IP_VS_SCTP_S_CLOSED 13
IP_VS_SCTP_S_ESTABLISHED 8
IP_VS_SCTP_S_SHUTDOWN_ACK_SENT 11
IP_VS_SCTP_S_SHUTDOWN_RECEIVED 10
IP_VS_SCTP_S_SHUTDOWN_SENT 9
IP_VS_TCP_S_CLOSE 6
IP_VS_TCP_S_CLOSE_WAIT 7
IP_VS_TCP_S_ESTABLISHED 1
IP_VS_TCP_S_FIN_WAIT 4
IP_VS_TCP_S_TIME_WAIT 5
IQM_CF_MIDTAP_IM__B 1
IQM_CF_MIDTAP_RE__B 0
IQM_CF_OUT_ENA_QAM__B 1
IQM_RC_RATE_OFS_LO__W 16
IREQ_ABORT_PATH_ACTIVE 6
IREQ_ACTIVE 3
IREQ_COMPLETE_IN_TARGET 0
IREQ_NO_AUTO_FREE_TAG 7
IREQ_PENDING_ABORT 4
IREQ_TC_ABORT_POSTED 5
IREQ_TERMINATED 1
IREQ_TMF 2
IRL_FC_SHIFT 24
IRQ_MODRT_RX_TIMER_SHIFT 16
IRQ_MODRT_TX_TIMER_SHIFT 0
```

```
IRQ_MOVE_CLEANUP_VECTOR 32
IRQTF_AFFINITY 2
IRQTF_FORCED_THREAD 3
IS_ANY_T0 9
IS_ANY_T1 10
IS_ATR_PRESENT 11
IS_ATR_VALID 12
IS_AUTOPPS_ACT 6
IS_BAD_CARD 16
IS_CMM_ABSENT 13
ISCSI_CMD_REQUEST_TYPE_SHIFT 14
ISCSI_KWQE_HEADER_LAYER_CODE_SHIFT 4
ISCSI_KWQE_INIT1_PAGE_SIZE_SHIFT 0
ISCSI_LOGIN_REQUEST_NUM_RESP_BDS_SHIFT 24
ISCSI_LOGIN_REQUEST_RESP_BUFFER_LENGTH_SHIFT 0
ISCSI_LOGIN_REQUEST_TYPE_SHIFT 14
ISCSI_LOGOUT_REQUEST_TYPE_SHIFT 14
ISCSI_SUSPEND_BIT 1
ISCSI_TEXT_REQUEST_NUM_RESP_BDS_SHIFT 24
ISCSI_TEXT_REQUEST_RESP_BUFFER_LENGTH_SHIFT 0
ISCSI_TEXT_REQUEST_TYPE_SHIFT 14
ISCSI_TMF_REQUEST_TYPE_SHIFT 14
ISDN_P_NT_E1 4
ISDN_P_NT_S0 2
ISDN_P_TE_E1 3
ISDN_P_TE_S0 1
IS_INVREV 8
ISP_ABORT_NEEDED 2
ISP_ABORT_RETRY 10
ISP_QUIESCE_NEEDED 20
ISP_UNRECOVERABLE 17
IT87_CFQ_SHIFT 3
IVTV_F_I_DECODING_YUV 12
IVTV_F_I_DEC_PAUSED 20
IVTV_F_I_DEC_YUV 7
IVTV_F_I_DMA 0
IVTV_F_I_EOS 4
IVTV_F_I_EV_DEC_STOPPED 28
IVTV_F_I_EV_VSYNC 29
IVTV_F_I_EV_VSYNC_ENABLED 31
IVTV_F_I_EV_VSYNC_FIELD 30
IVTV_F_I_FAILED 22
IVTV_F_I_HAVE_WORK 15
IVTV_F_I_PIO 19
IVTV_F_I_RADIO_USER 5
IVTV_F_I_UDMA 1
IVTV_F_I_UDMA_PENDING 2
IVTV_F_I_UPDATE_CC 9
IVTV_F_I_UPDATE_VPS 11
IVTV_F_I_UPDATE_WSS 10
IVTV_F_I_VALID_DEC_TIMINGS 14
IVTV_F_I_WORK_HANDLER_PCM 23
IVTV_F_I_WORK_HANDLER_PIO 18
IVTV_F_I_WORK_HANDLER_VBI 16
IVTV_F_I_WORK_HANDLER_YUV 17
IVTV_F_S_APPL_IO 8
IVTV_F_S_DMA_HAS_VBI 1
IVTV_F_S_DMA_PENDING 0
IVTV_F_S_INTERNAL_USE 5
IVTV_F_S_NEEDS_DATA 2
IVTV_F_S_PASSTHROUGH 6
IVTV_F_S_PIO_PENDING 9
IVTV_F_S_STREAMING 4
IVTV_F_S_STREAMOFF 7
IWCM_F_CALLBACK_DESTROY 1
IWCM_F_CONNECT_WAIT 2
```



```
IWLAGN_BT_FLAG_COEX_MODE_SHIFT 3
IWLAGN_EXT_BEACON_TIME_POS 22
IWL_FIRST_OFDM_RATE 4
IWL_MVM_STATUS_HW_CTKILL 1
IWL_MVM_STATUS_HW_RFKILL 0
IWL_MVM_STATUS_IN_HW_RESTART 3
IWL_MVM_STATUS_ROC_RUNNING 2
__IXGB_DOWN 0
IXGBE_ADVTXD_IDX_SHIFT 4
IXGBE_ADVTXD_L4LEN_SHIFT 8
IXGBE_ADVTXD_MACLEN_SHIFT 9
IXGBE_ADVTXD_MSS_SHIFT 16
IXGBE_ADVTXD_PAYLEN_SHIFT 14
__IXGBE_DOWN 2
IXGBE_DPMCS_MTSOS_SHIFT 16
IXGBE_EEPROM_RW_ADDR_SHIFT 2
IXGBE_EEPROM_RW_REG_DATA 16
IXGBE_ETQF_POOL_SHIFT 20
IXGBE_FCBUFF_BUFFCNT_SHIFT 8
IXGBE_FCBUFF_BUFFSIZE_SHIFT 3
IXGBE_FCBUFF_OFFSET_SHIFT 16
IXGBE_FCDMARW_LASTSIZE_SHIFT 16
__IXGBE_FCOE_TARGET 1
IXGBE_FDIRCMD_FLOW_TYPE_SHIFT 5
IXGBE_FDIRCMD_RX_QUEUE_SHIFT 16
IXGBE_FDIRCMD_VT_POOL_SHIFT 24
IXGBE_FDIRCTRL_DROP_Q_SHIFT 8
IXGBE_FDIRCTRL_FLEX_SHIFT 16
IXGBE_FDIRCTRL_FULL_THRESH_SHIFT 28
IXGBE_FDIRCTRL_MAX_LENGTH_SHIFT 24
IXGBE_FDIRHASH_SIG_SW_INDEX_SHIFT 16
IXGBE_MFLCN_RPFCE_SHIFT 4
IXGBE_MHADD_MFS_SHIFT 16
IXGBE_MSCA_DEV_TYPE_SHIFT 16
IXGBE_MSCA_NP_ADDR_SHIFT 0
IXGBE_MSCA_PHY_ADDR_SHIFT 21
IXGBE_RAH_VIND_SHIFT 18
__IXGBE_READ_I2C 5
IXGBE_RT2CR_MCL_SHIFT 12
IXGBE_RTRPT4C_BWG_SHIFT 9
IXGBE_RTRPT4C_MCL_SHIFT 12
IXGBE_RTRUP2TC_UP_SHIFT 3
IXGBE_RTTDT2C_BWG_SHIFT 9
IXGBE_RTTDT2C_MCL_SHIFT 12
IXGBE_RTTPCS_ARBD_SHIFT 22
IXGBE_RTTPT2C_BWG_SHIFT 9
IXGBE_RTTPT2C_MCL_SHIFT 12
__IXGBE_RX_CSUM_UDP_ZERO_ERR 5
__IXGBE_RX_FCOE 6
IXGBE_RXPBSIZE_SHIFT 10
IXGBE_SFF_VENDOR_OUI_BYTE0_SHIFT 24
IXGBE_SFF_VENDOR_OUI_BYTE1_SHIFT 16
IXGBE_SFF_VENDOR_OUI_BYTE2_SHIFT 8
IXGBE_SRRCTL_BSIZEHDRSIZE_SHIFT 2
IXGBE_TDPT2TCCR_BWG_SHIFT 9
IXGBE_TDPT2TCCR_MCL_SHIFT 12
IXGBE_TDTQ2TCCR_BWG_SHIFT 9
IXGBE_TDTQ2TCCR_MCL_SHIFT 12
__IXGBE_TESTING 0
__IXGBE_TX_FDIR_INIT_DONE 0
IXGBE_TX_FLAGS_VLAN_PRIO_SHIFT 29
IXGBE_TX_FLAGS_VLAN_SHIFT 16
__IXGBEVF_DOWN 2
__IXGBEVF_TESTING 0
IXGBE_VT_CTL_POOL_SHIFT 7
IXGBE_VT_MSGINFO_SHIFT 16
```

```
IXGB_MSCA_DEV_TYPE_SHIFT 16
IXGB_MSCA_NP_ADDR_SHIFT 0
IXGB_MSCA_PHY_ADDR_SHIFT 21
IXGB_RCTL_MO_SHIFT 12
J_ABORTED 3
JC42_CFG_HYST_SHIFT 9
__JI_COMMIT_RUNNING 0
JIFFIES_SHIFT 8
JME_FLAG_MSI 1
JME_FLAG_POLL 5
JME_FLAG_SHUTDOWN 6
JME_FLAG_SSET 2
JOB_AUTOWEP 7
JOB_DIE 0
JOB_EVENT 6
JOB_MIC 5
JOB_PROMISC 4
JOB_SCAN_RESULTS 9
JOB_STATS 3
JOB_WSTATS 8
JOB_XMIT 1
JOB_XMIT11 2
JSET_BITS 3
J_WRITERS_BLOCKED 1
J_WRITERS_QUEUED 2
KDB_DEBUG_FLAG_SHIFT 16
KEY_BRIGHTNESS_CYCLE 243
KEY_BRIGHTNESSDOWN 224
KEY_BRIGHTNESSUP 225
KEY_BRIGHTNESS_ZERO 244
KEY_DISPLAY_OFF 245
KEY_F14 184
KEY_F15 185
KEY_F16 186
KEY_F17 187
KEY_F18 188
KEY_FLAG_DEAD 1
KEY_FLAG_IN_QUOTA 3
KEY_FLAG_INSTANTIATED 0
KEY_FLAG_INVALIDATED 7
KEY_FLAG_NEGATIVE 5
KEY_FLAG_REVOKED 2
KEY_FLAG_ROOT_CAN_CLEAR 6
KEY_FLAG_USER_CONSTRUCT 4
KEY_GC_KEY_EXPIRED 0
KEY_GC_REAPING_KEYTYPE 2
KEY_GC_REAP_KEYTYPE 1
KEY_NUMLOCK 69
KEYP_LK_PTV_PTV_SHIFT 5
KEY_POWER 116
KEY_SLEEP 142
KEY_SWITCHVIDEOMODE 227
KEY_UNKNOWN 240
KEY_VIDEO_NEXT 241
KEY_VIDEO_PREV 242
KEY_VOLUMEDOWN 114
KEY_VOLUMEUP 115
KG_SHIFT 0
KMEM_ACCOUNTED_ACTIVATED 1
KMEM_ACCOUNTED_ACTIVE 0
KMEM_ACCOUNTED_DEAD 2
KPF_ANON 12
KPF_BUDDY 10
KPF_COMPOUND_HEAD 15
KPF_COMPOUND_TAIL 16
KPF_HUGE 17
```



```
KPF_KSM 21
KPF_MMAP 11
KPF_NOPAGE 20
KPF_THP 22
KP_ROW_SHIFT 4
KTHREAD_IS_PER_CPU 0
KTHREAD_SHOULD_PARK 2
KTHREAD_SHOULD_STOP 1
KVM_APIC_INIT 0
KVM_APIC_SIPI 1
KVM_FEATURE_ASYNC_PF 4
KVM_FEATURE_CLOCKSOURCE 0
KVM_FEATURE_CLOCKSOURCE2 3
KVM_FEATURE_CLOCKSOURCE_STABLE_BIT 24
KVM_FEATURE_NOP_IO_DELAY 1
KVM_FEATURE_PV_EOI 6
KVM_FEATURE_STEAL_TIME 5
KVM_IRQFD_RESAMPLE_IRQ_SOURCE_ID 1
KVM_MMU_HASH_SHIFT 10
KVM_REQ_CLOCK_UPDATE 8
KVM_REQ_MASTERCLOCK_UPDATE 18
KVM_USERSPACE_IRQ_SOURCE_ID 0
KWQE_LAYER_SHIFT 28
KWQE_OPCODE_SHIFT 16
L2BPERDMAP 13
L2CAP_CTRL_FINAL_SHIFT 7
L2CAP_CTRL_POLL_SHIFT 4
L2CAP_CTRL_REQSEQ_SHIFT 8
L2CAP_CTRL_SAR_SHIFT 14
L2CAP_CTRL_SUPER_SHIFT 2
L2CAP_CTRL_TXSEQ_SHIFT 1
L2CAP_EXT_CTRL_FINAL_SHIFT 1
L2CAP_EXT_CTRL_POLL_SHIFT 18
L2CAP_EXT_CTRL_REQSEQ_SHIFT 2
L2CAP_EXT_CTRL_SAR_SHIFT 16
L2CAP_EXT_CTRL_SUPER_SHIFT 16
L2CAP_EXT_CTRL_TXSEQ_SHIFT 18
L2_DB_SIZE 14
L2DBWORD 5
L2DISIZE 9
L2DTSLOTSIZE 5
L2EXTSPERSUM 5
L2INOSPEREXT 5
L2INOSPERIAG 12
L2LOGPSIZE 12
L2_QENTRY_SZ 12
L2XTSLOTSIZE 4
L3_CLS_PID_SHIFT 16
L3_CLS_TOSMASK_SHIFT 8
L3_CLS_TOS_SHIFT 0
L4_KWQ_CLOSE_REQ_LAYER_CODE_SHIFT 4
L4_KWQ_CONNECT_REQ1_LAYER_CODE_SHIFT 4
L4_KWQ_CONNECT_REQ2_LAYER_CODE_SHIFT 4
L4_KWQ_CONNECT_REQ3_LAYER_CODE_SHIFT 4
L4_KWQ_OFFLOAD_PG_LAYER_CODE_SHIFT 4
L4_KWQ_RESET_REQ_LAYER_CODE_SHIFT 4
L4_KWQ_UPDATE_PG_LAYER_CODE_SHIFT 4
L4_KWQ_UPLOAD_LAYER_CODE_SHIFT 4
LAS_BIT_BIGENDIAN 24
LC_LINK_WIDTH_SHIFT 0
LC_SPEED_CHANGE_ATTEMPTS_ALLOWED_SHIFT 3
LDO_I2C_EN_SHIFT 0
LDO_SEL_SHIFT 2
LDO_SEQ_SHIFT 2
LED_CAPSL 1
LED_COMPOSE 3
```

```
LED_KANA 4
LED_LINK_SHIFT 12
LED_NUML 0
LED_SCROLLL 2
LED_TXRX_SHIFT 8
LFA_LINK_FLAP_REASON_OFFSET 0
LINE6_INDEX_PAUSE_PLAYBACK 12
LINK_FLAP_AVOIDANCE_COUNT_OFFSET 8
LINK_FLAP_COUNT_OFFSET 16
__LINK_STATE_DORMANT 4
__LINK_STATE_NOCARRIER 2
__LINK_STATE_PRESENT 1
__LINK_STATE_START 0
LK_STATE_IN_USE 0
LM3533_ALS_FLAG_INT_ENABLED 1
LM3533_BOOST_FREQ_SHIFT 0
LM3533_BOOST_OVP_SHIFT 1
LM73_CTRL_RES_SHIFT 5
LmRAMPAGE_LSHIFT 8
LOCAL_ADDRESSING_MODE 11
LOCAL_BURST_ENABLE 8
LOCAL_BUS_WIDTH 0
LOCAL_DMA_CHANNEL_0_INTERRUPT_ENABLE 18
LOCAL_INTERRUPT_INPUT_ENABLE 11
LOCAL_LOOP_UPDATE 6
LOCAL_OUT_ZLP 6
LOG2_LOG_TABLE_SIZE 5
log_FLUSH 4
log_INLINELOG 1
log_QUIESCE 3
log_SYNCBARRIER 2
LOOPBACK_NONE 0
LOOPBACK_XGMII 3
LOOP_RESYNC_NEEDED 4
LowThresholdShift 12
LOWWAIT 2
LP8725_BUCK_CL_S 6
LP8727_ICHG_SHIFT 4
LP8788_ALM_EN_S 7
LP8788_BL_DIM_MODE_SHIFT 1
LP8788_BL_FULLSCALE_SHIFT 2
LP8788_BL_PWM_POLARITY_SHIFT 6
LP8788_BL_RAMP_RISE_SHIFT 4
LPFC_DATA_READY 1
LPFC_ELS_RING 2
LPFC_FCP_RING 0
LPFC_FIP_ELS_ID_SHIFT 14
LP_PREEMPT_REQUEST 1
LSFL_CB_DELAY 9
LSFL_NODIR 10
LSFL_RCOM_READY 5
LSFL_RCOM_WAIT 6
LSFL_RECOVER_DOWN 1
LSFL_RECOVER_LOCK 2
LSFL_RECOVER_STOP 0
LSFL_RECOVER_WORK 3
LSFL_RUNNING 4
LSFL_TIMEWARN 8
LSFL_UEVENT_WAIT 7
LVDS_BL_MOD_LEVEL_SHIFT 8
LZMA_LEN_NUM_LOW_BITS 3
LZMA_LEN_NUM_MID_BITS 3
LZMA_NUM_POS_BITS_MAX 4
LZMA_NUM_POS_SLOT_BITS 6
M8051_RESET 0
M98088_MICPRE_SHIFT 5
```



```
M98090_ADCHP_SHIFT 0
M98090_DACHP_SHIFT 0
M98090_DMIC_COMP_SHIFT 4
M98090_FREQ_SHIFT 4
M98090_IJDET_SHIFT 2
M98090_MICCLK_SHIFT 4
M98090_MIC_PA1EN_SHIFT 5
M98090_MIC_PA2EN_SHIFT 5
M98090_PERFMODE_SHIFT 1
M98090_TDM_SLOTDLY_SHIFT 0
M98090_TDM_SLOTL_SHIFT 6
M98090_TDM_SLOTR_SHIFT 4
M98090_USE_M1_SHIFT 0
M98095_MICPRE_SHIFT 5
MAC_ADDR_IDX_SHIFT 4
MAC_ADDR_TYPE_SHIFT 16
MAC_ADDR_VALID 0
MACB_REV_SIZE 16
MAC_CFG_DESC_TX_1_DST_INDEX_POS 16
MAC_CFG_DESC_TX_2_L2_TRANSLATION_TYPE_POS 18
MAC_CFG_DESC_TX_2_NUM_OF_DESCRIPTORS_POS 0
MAC_CTRL_HALF_LEFT_BUF_SHIFT 28
MAC_CTRL_PRMLEN_SHIFT 10
MAC_CTRL_SPEED_SHIFT 20
MAC_FLOW_CTRL_PT_SHIFT 16
MAC_HALF_DUPLX_CTRL_ABEBT_SHIFT 20
MAC_HALF_DUPLX_CTRL_JAMIPG_SHIFT 24
MAC_HALF_DUPLX_CTRL_RETRY_SHIFT 12
MAC_IPG_IFG_IPGR1_SHIFT 16
MAC_IPG_IFG_IPGR2_SHIFT 24
MAC_IPG_IFG_IPGT_SHIFT 0
MAC_IPG_IFG_MIFG_SHIFT 8
Magic 1
MAP_ISR_S2_CST 6
MAP_ISR_S2_GTT 6
MAX77686_OPMODE_BUCK234_SHIFT 4
MAX77686_OPMODE_SHIFT 6
MAX8925_IRQ_RTC_ALARM0 22
MAX8997_LED_FLASH_SHIFT 3
MAX8997_LED_MOVIE_SHIFT 4
MAX9877_OSC_OFFSET 4
MAX_HWIFS 10
MAX_ORDER 11
MAX_PFC_PRIORITIES 8
MAX_READ_REQUEST_SHIFT 12
MAX_READ_REQUEST_SZ_SHIFT 12
MAX_SWAPFILES_SHIFT 5
M_BITS 9
MB_RFD_PROD_INDX_SHIFT 0
MB_RRD_CONS_INDX_SHIFT 11
MB_TPD_PROD_INDX_SHIFT 22
MBX_INTR_WAIT 2
MBX_UPDATE_FLASH_ACTIVE 3
MC13783_ADC1_ATO_SHIFT 11
MC13783_POWER_CONTROL_2_ON1BDBNC 4
MC13783_POWER_CONTROL_2_ON2BDBNC 6
MC13783_POWER_CONTROL_2_ON3BDBNC 8
MC13XXX_ADC1_CHAN0_SHIFT 5
MC13XXX_ADC1_CHAN1_SHIFT 8
MC_CHP_IO_CNTL_A1__MEM_SYNC_ENA__SHIFT 24
MC_CHP_IO_CNTL_B1__MEM_SYNC_ENB__SHIFT 24
MC_CMD_CAPABILITIES_TURBO_ACTIVE_LBN 2
MC_CMD_GET_LINK_OUT_FULL_DUPLEX_LBN 1
MC_CMD_GET_LINK_OUT_LINK_UP_LBN 0
MC_CMD_GET_PHY_CFG_OUT_BIST_CABLE_LONG_LBN 2
MC_CMD_GET_PHY_CFG_OUT_BIST_CABLE_SHORT_LBN 1
```

```
MC_CMD_GET_PHY_CFG_OUT_BIST_LBN 6
MC_CMD_GET_PHY_CFG_OUT_LOWPOWER_LBN 3
MC_CMD_GET_PHY_CFG_OUT_POWEROFF_LBN 4
MC_CMD_GET_PHY_CFG_OUT_TXDIS_LBN 5
MC_CMD_MMD_CLAUSE22 0
MC_CMD_NVRAM_INFO_OUT_PROTECTED_LBN 0
MC_CMD_PHY_CAP_10000FDX_LBN 7
MC_CMD_PHY_CAP_1000FDX_LBN 6
MC_CMD_PHY_CAP_1000HDX_LBN 5
MC_CMD_PHY_CAP_100FDX_LBN 4
MC_CMD_PHY_CAP_100HDX_LBN 3
MC_CMD_PHY_CAP_10FDX_LBN 2
MC_CMD_PHY_CAP_10HDX_LBN 1
MC_CMD_PHY_CAP_AN_LBN 10
MC_CMD_PHY_CAP_ASYM_LBN 9
MC_CMD_PHY_CAP_PAUSE_LBN 8
MC_CMD_SET_LINK_IN_LOWPOWER_LBN 0
MC_CMD_SET_LINK_IN_POWEROFF_LBN 1
MC_CMD_SET_LINK_IN_TXDIS_LBN 2
MC_CMD_SET_MAC_IN_REJECT_UNCST_LBN 0
MCC_WRB_SGE_CNT_SHIFT 3
MC_DISP0R_INIT_LAT_SHIFT 8
MC_DISP1R_INIT_LAT_SHIFT 12
MCE_OVERFLOW 0
MC_HASH_SZ_LOG 9
MCH_PRD_LEN_SHIFT 16
MCH_SSP_FR_TYPE_SHIFT 13
MC_NANOSECOND_BITS 30
MCPR_IMC_COMMAND_OPERATION_BITSHIFT 28
MCPR_IMC_COMMAND_TRANSFER_ADDRESS_BITSHIFT 8
MCS_SEL_IN_FREQ_SHIFT 4
MC_VECTOR 18
MD_ARRAY_FIRST_USE 3
MD_CHANGE_CLEAN 1
MD_CHANGE_DEVS 0
MD_CHANGE_PENDING 2
MD_DISK_ACTIVE 1
MD_DISK_FAULTY 0
MD_DISK_REMOVED 3
MD_DISK_SYNC 2
MD_DISK_WRITEMOSTLY 9
MDIO_CLK_SEL_SHIFT 24
MDIO_DATA_OP_SHIFT 28
MDIO_DATA_PMD_SHIFT 23
MDIO_DATA_RA_SHIFT 18
MDIO_DATA_SHIFT 0
MDIO_DATA_TA_SHIFT 16
MDIO_MMD_PHYXS 4
MDIO_REG_ADDR_SHIFT 16
MDIO_TX0_TX_DRIVER_PREEMPHASIS_SHIFT 12
MD_NO_FUA 7
MdpMinorShift 6
MD_RECOVERY_CHECK 7
MD_RECOVERY_DONE 4
MD_RECOVERY_ERROR 10
MD_RECOVERY_FROZEN 9
MD_RECOVERY_INTR 3
MD_RECOVERY_NEEDED 5
MD_RECOVERY_RECOVER 2
MD_RECOVERY_REQUESTED 6
MD_RECOVERY_RESHAPE 8
MD_RECOVERY_RUNNING 0
MD_RECOVERY_SYNC 1
MD_SB_BITMAP_PRESENT 8
MD_SB_CLEAN 0
MEDIA_FLAG_REGISTERED 0
```



```
MEGASAS_FUSION_IN_RESET 0
MEGASAS_REQ_DESCRIPT_FLAGS_TYPE_SHIFT 1
MEI_CL_EVENT_RX 0
MEMCG_SOCK_ACTIVE 0
MEMCTL_CMD_SHIFT 13
MEMCTL_FREQ_SHIFT 8
MEMORY_SPACE_LOCAL_BUS_WIDTH 0
MEM_SDRAM_MODE_REG__MEM_MODE_REG__SHIFT 0
MESH_IFACE_BIT_OFFSET 12
MESH_WORK_DRIFT_ADJUST 4
MESH_WORK_GROW_MPATH_TABLE 1
MESH_WORK_GROW_MPP_TABLE 2
MESH_WORK_HOUSEKEEPING 0
MESH_WORK_MBSS_CHANGED 5
MESH_WORK_ROOT 3
META_discard 4
META_forcewrite 5
META_free 1
META_io 6
MI_COM_PHY_ADDR_SHIFT 21
MI_COM_REG_ADDR_SHIFT 16
MICRO_SELECT_PREAMPLI_OFFSET 2
MID_DMA_CHAN_SHIFT 6
MID_DMA_COUNT_SHIFT 16
MID_DMA_VCI_SHIFT 6
MID_RED_SHIFT 25
MID_SEG_ID_SHIFT 28
MID_SEG_PR_SHIFT 25
MID_SEG_RATE_SHIFT 19
MID_SEG_VCI_SHIFT 4
MID_VCI_LOCATION_SHIFT 18
MID_VCI_MODE_SHIFT 30
MID_VCI_SIZE_SHIFT 15
MIIpmdShift 7
MIIregShift 2
MII_TG3_AUXCTL_MISC_RDSEL_SHIFT 12
MIN_OCQP_SHIFT 12
MINORBITS 20
MIN_PBL_SHIFT 8
MIN_RQT_SHIFT 10
MISC_REGISTERS_GPIO_3 3
MISC_REGISTERS_GPIO_CLR_POS 16
MISC_REGISTERS_GPIO_FLOAT_POS 24
MISC_REGISTERS_GPIO_INT_CLR_POS 24
MISC_REGISTERS_GPIO_INT_SET_POS 16
MISC_REGISTERS_GPIO_SET_POS 8
MISC_SPIO_CLR_POS 16
MISC_SPIO_FLOAT_POS 24
MISC_SPIO_INT_OLD_SET_POS 16
MISC_SPIO_SET_POS 8
MI_STORE_DWORD_INDEX_SHIFT 2
MIXART_MOTHERBOARD_ELF_INDEX 1
MIXART_NOTIFY_CARD_OFFSET 12
MIXART_NOTIFY_PCM_OFFSET 8
ML_BITS 4
MLX4_CMPT_SHIFT 24
MLX4_EVENT_TYPE_PORT_MNG_CHG_EVENT 29
MLX4_RSS_QPC_FLAG_OFFSET 13
MLX4_UPD_QP_PATH_MASK_ETH_RX_BLOCK_1P 43
MLX4_UPD_QP_PATH_MASK_ETH_RX_BLOCK_TAGGED 44
MLX4_UPD_QP_PATH_MASK_ETH_RX_BLOCK_UNTAGGED 42
MLX4_UPD_QP_PATH_MASK_ETH_TX_BLOCK_1P 40
MLX4_UPD_QP_PATH_MASK_ETH_TX_BLOCK_TAGGED 41
MLX4_UPD_QP_PATH_MASK_ETH_TX_BLOCK_UNTAGGED 39
MLX4_UPD_QP_PATH_MASK_SCHED_QUEUE 46
MLX4_UPD_QP_PATH_MASK_VLAN_INDEX 37
```

```
MLX5_CMD_DATA 0
MLX5_EVENT_TYPE_CMD 10
MLX5_EVENT_TYPE_PAGE_REQUEST 11
MLX5_IB_MMAP_CMD_SHIFT 8
MLX5_IB_SQ_STRIDE 6
MMF_DUMPABLE 0
MMF_DUMP_FILTER_SHIFT 2
MMF_DUMP_SECURELY 1
MMF_HAS_UPROBES 19
MMF_RECALC_UPROBES 20
MMF_VM_MERGEABLE 16
MMIO_GEN_LOW_SHIFT 9
MMIO_SPTE_GEN_HIGH_SHIFT 52
MMIO_SPTE_GEN_LOW_SHIFT 3
MM_SHIFT 6
MODEL24_SHIFT 1
MODE_SHIFT 5
MPI_NIC_FUNCTION_SHIFT 6
MPI_RESET_NEEDED 19
MP_ISA_BUS 0
MPU3050_PWR_MGM_POS 6
MPU401_MODE_BIT_INPUT 0
MPU401_MODE_BIT_OUTPUT 1
MPU401_MODE_BIT_OUTPUT_TRIGGER 3
MPX_MODE 3
MR_COMPLETE 2
MR_INPROGRESS 1
MR_RAID_CTX_RAID_FLAGS_IO_SUB_TYPE_SHIFT 4
MR_START 0
MSBD0_X_SHIFT 28
MSBD0_Y_SHIFT 24
MSBD1_SHIFT 24
MSC_RAW 3
MSC_SCAN 4
MSI_ADDR_DEST_MODE_SHIFT 2
MSI_DATA_TRIGGER_SHIFT 15
MSPRO_BLOCK_PART_SHIFT 3
MSR_C_LO_VID_SHIFT 8
MSR_LINK_SHIFT 0
MSRPM_ALLOC_ORDER 1
MSSShift 16
MS_X0_SHIFT 0
MS_X1_SHIFT 8
MS_X2_SHIFT 16
MS_X3_SHIFT 0
MS_X4_SHIFT 8
MS_X5_SHIFT 16
MS_Y0_SHIFT 4
MS_Y1_SHIFT 12
MS_Y2_SHIFT 20
MS_Y3_SHIFT 4
MS_Y4_SHIFT 12
MS_Y5_SHIFT 20
MTIP_ABAR 5
MTIP_DDF_CLEANUP_BIT 5
MTIP_DDF_INIT_DONE_BIT 7
MTIP_DDF_OVER_TEMP_BIT 2
MTIP_DDF_REBUILD_FAILED_BIT 8
MTIP_DDF_REMOVE_PENDING_BIT 1
MTIP_DDF_RESUME_BIT 6
MTIP_DDF_SEC_LOCK_BIT 0
MTIP_DDF_WRITE_PROTECT_BIT 3
MTIP_PF_DM_ACTIVE_BIT 3
MTIP_PF_EH_ACTIVE_BIT 1
MTIP_PF_IC_ACTIVE_BIT 0
MTIP_PF_ISSUE_CMDS_BIT 5
```



```
MTIP_PF_REBUILD_BIT 6
MTIP_PF_SE_ACTIVE_BIT 2
MTIP_PF_SVC_THD_ACTIVE_BIT 4
MTIP_PF_SVC_THD_STOP_BIT 8
MTIP_TAG_INTERNAL 0
MT_ST_BLKSIZE_SHIFT 0
MT_ST_DENSITY_SHIFT 24
MT_ST_SOFTERR_SHIFT 0
MTYPE_OFS 4
MUSB_DEVCTL_VBUS_SHIFT 3
MUSYCC_CCD_BUFFER_LENGTH 16
MUSYCC_CCD_BUFFER_LOC 24
MUSYCC_CCD_MAX_LENGTH 10
MUSYCC_CCD_PROTO_SHIFT 12
MVMDIO_SMI_DATA_SHIFT 0
MVMDIO_SMI_PHY_ADDR_SHIFT 16
MVMDIO_SMI_PHY_REG_SHIFT 21
MV_PRIMARY_BAR 0
MV_U3D_EPXCR_EP_ENABLE_SHIFT 4
MV_U3D_EPXCR_MAX_BURST_SIZE_SHIFT 12
MV_U3D_EPXCR_MAX_PACKET_SIZE_SHIFT 16
MWAIT_SUBSTATE_SIZE 4
NAK_OUT_PACKETS 4
NAK_OUT_PACKETS 5
NAK_OUT_PACKETS_MODE 2
NAME_HASH_SHIFT 5
NAPI_STATE_NPSVC 2
NAPI_STATE_SCHED 0
NCI_DATA_EXCHANGE_TO 3
NCI_INIT 0
NCI_SPI_ACK_SHIFT 6
NCI_UP 1
NES_CQP_APBVT_NIC_SHIFT 16
NES_CQP_OP_LOGICAL_PORT_SHIFT 26
NES_CQP_OP_TERMLEN_SHIFT 28
NET2272_PCI_IRQ 0
NET2272_RESET 0
NET_CONGESTED 0
NETDEV_MCAST_ALL_OFF 11
NETDEV_MCAST_ALL_ON 10
NETLBL_DOMHSH_BITSIZE 7
NETQ_STOPPED 3
NEW_CUR_UUID 17
NF_ARP_NUMHOOKS 3
NF_BR_BROUTING 5
NF_BR_LOCAL_OUT 3
NF_BR_NUMHOOKS 6
NF_BR_PRE_ROUTING 0
NFC_HCI_QUIRK_SHORT_CLEAR 0
NFC_PROTO_NFC_DEP 5
NFCWILINK_FW_DOWNLOAD 1
NF_INET_FORWARD 2
NF_INET_LOCAL_IN 1
NF_INET_LOCAL_OUT 3
NF_INET_NUMHOOKS 5
NF_INET_POST_ROUTING 4
NF_INET_PRE_ROUTING 0
NFS4CLNT_BIND_CONN_TO_SESSION 10
NFS4CLNT_CHECK_LEASE 1
NFS4CLNT_DELEGRETURN 5
NFS4CLNT_LEASE_CONFIRM 7
NFS4CLNT_LEASE_EXPIRED 2
NFS4CLNT_PURGE_STATE 9
NFS4CLNT_RECLAIM_NOGRACE 4
NFS4CLNT_RECLAIM_REBOOT 3
NFS4CLNT_SERVER_SCOPE_MISMATCH 8
```

```
NFS4CLNT_SESSION_RESET 6
NFS4_SESSION_INITING 0
NFS4_SLOT_TBL_DRAINING 0
NFS_CONTEXT_BAD 2
NFS_CONTEXT_ERROR_WRITE 0
NFS_CONTEXT_RESEND_WRITES 1
NFS_CS_CHECK_LEASE_TIME 5
NFS_CS_MIGRATION 2
NFS_CS_NORESVPORT 0
NFS_CS_RENEWD 3
NFS_CS_STOP_RENEW 4
NFSD4_CLIENT_CB_KILL 1
NFSD4_CLIENT_CB_UPDATE 0
NFSD4_CLIENT_CONFIRMED 4
NFSD4_CLIENT_STABLE 2
NFS_DELEGATED_STATE 1
NFS_DELEGATION_NEED_RECLAIM 0
NFS_DELEGATION_REFERENCED 3
NFS_DELEGATION_RETURN 1
NFS_DELEGATION_RETURN_IF_CLOSED 2
NFS_DELEGATION_RETURNING 4
NFS_DEVICEID_INVALID 0
NFS_DEVICEID_UNAVAILABLE 1
NFS_INO_ADVISE_RDPLUS 0
NFS_INO_FSCACHE 5
NFS_INO_STALE 1
NFS_IOHDR_EOF 1
NFS_IOHDR_ERROR 0
NFS_IOHDR_NEED_RESCHED 4
NFS_IOHDR_REDO 2
NFS_IO_INPROGRESS 0
NFS_LAYOUT_BULK_RECALL 2
NFS_LAYOUT_RETURN 4
NFS_LAYOUT_ROC 3
NFS_LOCK_INITIALIZED 0
NFS_LSEG_ROC 1
NFS_LSEG_VALID 0
NFS_OPEN_STATE 2
NFS_O_RDONLY_STATE 3
NFS_O_RDWR_STATE 5
NFS_OWNER_RECLAIM_NOGRACE 1
NFS_OWNER_RECLAIM_REBOOT 0
NFS_O_WRONLY_STATE 4
NFS_STATE_POSIX_LOCKS 8
NFS_STATE_RECLAIM_NOGRACE 7
NFS_STATE_RECLAIM_REBOOT 6
NFS_STATE_RECOVERY_FAILED 9
NHMEX_M_PMON_CTL_INC_SEL_SHIFT 9
NHMEX_R_PMON_CTL_EV_SEL_SHIFT 1
NIG_LATCH_BC_ENABLE_MI_INT 0
NIG_STATUS_XGXS0_LINK_STATUS_SIZE 18
NILFS_I_BMAP 7
NILFS_I_BUSY 3
NILFS_I_COLLECTED 4
NILFS_I_DIRTY 1
NILFS_I_GCINODE 8
NILFS_I_INODE_DIRTY 6
NILFS_I_NEW 0
NILFS_I_QUEUED 2
NILFS_I_UPDATED 5
NILFS_SC_DIRTY 0
NILFS_SC_HAVE_DELTA 4
NILFS_SC_PRIOR_FLUSH 3
NILFS_SC_SUPER_ROOT 2
NILFS_SC_UNCLOSED 1
NILFS_SEGMENT_USAGE_ACTIVE 0
```



```
NILFS_SEGMENT_USAGE_ERROR 2
NM_VECTOR 7
NO_DECELERATION 1
NO_FILTER 5
NOHZ_TICK_STOPPED 0
NOSV 7
NOT_MASKED_PD_BITS 17
NPIV_CONFIG_NEEDED 16
NR_FSCACHE_OBJECT_EVENTS 7
NR_MOVE_TYPE 2
NR_PAGEFLAGS 25
NR_SHAPERS 16
NTP_SCALE_SHIFT 32
NUM_MAC_INDEX_DRIVER 4
NUM_NL80211_IFTYPES 11
NUM_TSF_IDS 4
NUM_TSSI_FRAMES 4
NUM_VCI_BITS 10
NUM_VPI_BITS 0
NVDEV_ENGINE_DEVICE 0
NVDEV_ENGINE_DMAOBJ 18
NVM_COMB_VER_SHFT 8
NVME_CC_MPS_SHIFT 7
NVM_ETRACK_SHIFT 16
NV_MMIO_BAR 5
NVREG_ADAPTCTL_PHYSHIFT 24
NVREG_BKOFFCTRL_GEAR 12
NVREG_BKOFFCTRL_SELECT 24
NVREG_MIICTL_ADDRSHIFT 5
NVREG_RINGSZ_RXSHIFT 16
NVREG_RINGSZ_TXSHIFT 0
NV_TX2_TSO_SHIFT 14
NWD_SHIFT 4
__NX_DEV_UP 1
OBJ_REQ_EXISTS 3
OBJ_REQ_KNOWN 2
OCFS2_HASH_SHIFT 5
OCRDMA_ALLOC_LKEY_FMR_SHIFT 1
OCRDMA_ALLOC_LKEY_LOCAL_WR_SHIFT 5
OCRDMA_ALLOC_LKEY_PBL_SIZE_SHIFT 16
OCRDMA_ALLOC_LKEY_REMOTE_ATOMIC_SHIFT 6
OCRDMA_ALLOC_LKEY_REMOTE_RD_SHIFT 4
OCRDMA_ALLOC_LKEY_REMOTE_WR_SHIFT 3
OCRDMA_CREATE_AH_ENTRY_SIZE_SHIFT 23
OCRDMA_CREATE_AH_NUM_PAGES_SHIFT 19
OCRDMA_CREATE_AH_PAGE_SIZE_SHIFT 16
OCRDMA_CREATE_CQ_CNT_SHIFT 27
OCRDMA_CREATE_CQ_EQID_SHIFT 22
OCRDMA_CREATE_CQ_PAGE_SIZE_SHIFT 16
OCRDMA_CREATE_CQ_TYPE_SHIFT 24
OCRDMA_CREATE_EQ_CNT_SHIFT 26
OCRDMA_CREATE_MQ_CQ_ID_SHIFT 16
OCRDMA_CREATE_MQ_RING_SIZE_SHIFT 16
OCRDMA_CREATE_QP_REQ_DPP_CREDIT_SHIFT 16
OCRDMA_CREATE_QP_REQ_MAX_IRD_SHIFT 0
OCRDMA_CREATE_QP_REQ_MAX_ORD_SHIFT 16
OCRDMA_CREATE_QP_REQ_MAX_RQE_SHIFT 0
OCRDMA_CREATE_QP_REQ_MAX_SGE_RECV_SHIFT 16
OCRDMA_CREATE_QP_REQ_MAX_SGE_SEND_SHIFT 16
OCRDMA_CREATE_QP_REQ_MAX_SGE_WRITE_SHIFT 0
OCRDMA_CREATE_QP_REQ_MAX_WQE_SHIFT 16
OCRDMA_CREATE_QP_REQ_NUM_RQ_PAGES_SHIFT 0
OCRDMA_CREATE_QP_REQ_NUM_WQ_PAGES_SHIFT 16
OCRDMA_CREATE_QP_REQ_PD_ID_SHIFT 0
OCRDMA_CREATE_QP_REQ_QPT_SHIFT 29
OCRDMA_CREATE_QP_REQ_RQ_CQID_SHIFT 0
```

```
OCRDMA_CREATE_QP_REQ_RQE_SIZE_SHIFT 0
OCRDMA_CREATE_QP_REQ_RQ_PAGE_SIZE_SHIFT 19
OCRDMA_CREATE_QP_REQ_SQ_PAGE_SIZE_SHIFT 16
OCRDMA_CREATE_QP_REQ_WQ_CQID_SHIFT 16
OCRDMA_CREATE_QP_REQ_WQE_SIZE_SHIFT 16
OCRDMA_CREATE_SRQ_MAX_SGE_RECV_SHIFT 16
OCRDMA_CREATE_SRQ_NUM_RQ_PAGES_SHIFT 16
OCRDMA_CREATE_SRQ_PG_SZ_SHIFT 16
OCRDMA_CREATE_SRQ_RQE_SIZE_SHIFT 0
OCRDMA_DB_CQ_RING_ID_EXT_MASK_SHIFT 1
OCRDMA_DESTROY_CQ_QID_SHIFT 0
OCRDMA_MCH_SUBSYS_SHIFT 8
OCRDMA_MODIFY_SRQ_LIMIT_SHIFT 16
OCRDMA_MQE_HDR_EMB_SHIFT 0
OCRDMA_QP_PARAMS_ACK_TIMEOUT_SHIFT 27
OCRDMA_QP_PARAMS_HOP_LMT_SHIFT 24
OCRDMA_QP_PARAMS_MAX_ORD_SHIFT 16
OCRDMA_QP_PARAMS_PATH_MTU_SHIFT 18
OCRDMA_QP_PARAMS_RETRY_CNT_SHIFT 24
OCRDMA_QP_PARAMS_RNR_NAK_TIMER_SHIFT 27
OCRDMA_QP_PARAMS_RNR_RETRY_CNT_SHIFT 24
OCRDMA_QP_PARAMS_STATE_SHIFT 5
OCRDMA_QP_PARAMS_TCLASS_SHIFT 24
OCRDMA_QP_PARAMS_VLAN_SHIFT 16
OCRDMA_REG_NSMR_BIND_MEMWIN_SHIFT 24
OCRDMA_REG_NSMR_CONT_LAST_SHIFT 31
OCRDMA_REG_NSMR_CONT_NUM_PBL_SHIFT 16
OCRDMA_REG_NSMR_HPAGE_SIZE_SHIFT 16
OCRDMA_REG_NSMR_LAST_SHIFT 31
OCRDMA_REG_NSMR_LOCAL_WR_SHIFT 29
OCRDMA_REG_NSMR_NUM_PBL_SHIFT 16
OCRDMA_REG_NSMR_REMOTE_ATOMIC_SHIFT 30
OCRDMA_REG_NSMR_REMOTE_RD_SHIFT 28
OCRDMA_REG_NSMR_REMOTE_WR_SHIFT 27
OCRDMA_WQE_FLAGS_SHIFT 5
OCRDMA_WQE_OPCODE_SHIFT 0
OCRDMA_WQE_SIZE_SHIFT 18
OCRDMA_WQE_TYPE_SHIFT 16
OFDM_SC_RA_RAM_ECHO_THRES_2K__B 8
OFDM_SC_RA_RAM_ECHO_THRES_8K__B 0
OMAP_CTRL_USB_PWRCTL_CLK_CMD_SHIFT 14
OMAP_CTRL_USB_PWRCTL_CLK_FREQ_SHIFT 22
ONETOUCH_BUTTON 148
OO_SHIFT 16
OP_ANI_RUNNING 32
OP_BT_PRIORITY_DETECTED 8
OP_BT_SCAN 16
OP_ENABLE_BEACON 4
OP_INVALID 1
OP_SCANNING 2
OP_TSF_RESET 64
OSD_OFFSET_MAX_BITS 28
OSST_FRAME_SHIFT 6
OSST_SECTOR_SHIFT 9
OVS_KEY_ATTR_ARP 13
OVS_KEY_ATTR_ENCAP 1
OVS_KEY_ATTR_ETHERNET 4
OVS_KEY_ATTR_ETHERTYPE 6
OVS_KEY_ATTR_ICMP 11
OVS_KEY_ATTR_ICMPV6 12
OVS_KEY_ATTR_IN_PORT 3
OVS_KEY_ATTR_IPV4 7
OVS_KEY_ATTR_IPV6 8
OVS_KEY_ATTR_ND 14
OVS_KEY_ATTR_PRIORITY 2
OVS_KEY_ATTR_SKB_MARK 15
```



```
OVS_KEY_ATTR_TCP 9
OVS_KEY_ATTR_TUNNEL 16
OVS_KEY_ATTR_UDP 10
OVS_KEY_ATTR_VLAN 5
OXU_REV_SHIFT 16
OXYGEN_AC97_REG_ADDR_SHIFT 16
OXYGEN_AC97_REG_CODEC_SHIFT 24
OXYGEN_A_MONITOR_ROUTE_0_SHIFT 0
OXYGEN_A_MONITOR_ROUTE_1_SHIFT 2
OXYGEN_A_MONITOR_ROUTE_2_SHIFT 4
OXYGEN_A_MONITOR_ROUTE_3_SHIFT 6
OXYGEN_MULTICH_FORMAT_SHIFT 2
OXYGEN_PLAY_DAC0_SOURCE_SHIFT 8
OXYGEN_PLAY_DAC1_SOURCE_SHIFT 10
OXYGEN_PLAY_DAC2_SOURCE_SHIFT 12
OXYGEN_PLAY_DAC3_SOURCE_SHIFT 14
OXYGEN_REC_FORMAT_A_SHIFT 0
OXYGEN_REC_FORMAT_B_SHIFT 2
OXYGEN_REC_FORMAT_C_SHIFT 4
OXYGEN_SPDIF_CATEGORY_SHIFT 4
OXYGEN_SPDIF_CS_RATE_SHIFT 12
OXYGEN_SPDIF_FORMAT_SHIFT 0
OXYGEN_SPDIF_OUT_RATE_SHIFT 24
OXYGEN_SPI_CODEC_SHIFT 4
OZ_VERSION_SHIFT 2
P0_EN_1V0X_LBN 0
P0_EN_1V2_LBN 1
P0_EN_2V5_LBN 2
P0_EN_3V3X_LBN 3
P0_EN_5V_LBN 4
P0_X_TRST_LBN 6
P1_AFE_PWD_LBN 0
P1_DSP_PWD25_LBN 1
P1_SPARE_LBN 4
PACKET_LRA_VALID 3
PACKET_MERGE_SEGS 4
PACKET_NWA_VALID 2
PACKET_WRITABLE 1
PA_CTRL_APALP 4
PA_CTRL_APAPRECH 5
PAGE_ALLOC_COSTLY_ORDER 3
PAGE_CACHE_SHIFT 12
PAGE_SHIFT 12
PALMAS_PRIMARY_SECONDARY_PAD1_GPIO_1_SHIFT 3
PALMAS_PRIMARY_SECONDARY_PAD1_GPIO_2_SHIFT 5
PALMAS_SMPS10_CTRL_MODE_SLEEP_SHIFT 4
PALMAS_SMPS12_CTRL_MODE_SLEEP_SHIFT 2
PANEL_LIGHT_OFF_DELAY_SHIFT 0
PANEL_LIGHT_ON_DELAY_SHIFT 0
PANEL_POWER_CYCLE_DELAY_SHIFT 0
PANEL_POWER_DOWN_DELAY_SHIFT 16
PANEL_POWER_UP_DELAY_SHIFT 16
PART_BITS 4
PARTN_BITS 6
PASS_ACCEPT_REQ 4
PASS_ESTAB 5
PBC_PORT_SEL_LSB 26
PBC_VL_NUM_LSB 27
PCAP_ADC_ADA1_SHIFT 4
PCAP_ADC_ADA2_SHIFT 7
PCAP_ADC_TS_M_SHIFT 17
PCAP_IRQ_ADCDONE2 20
PCAP_REGISTER_ADDRESS_SHIFT 26
PCCRX_SHIFT 8
PCCRXTO_SHIFT 16
PCCTX_SHIFT 8
```

```
PCCTXTO_SHIFT 16
PCH_BIT_SJW_SHIFT 6
PCH_BIT_TSEG1_SHIFT 8
PCH_BIT_TSEG2_SHIFT 12
PCH_GBE_MIIM_PHY_ADDR_SHIFT 21
PCH_GBE_MIIM_REG_ADDR_SHIFT 16
PCH_GBE_PCI_BAR 1
PCI_BASE2_RANGE 16
_PCIB_op_pending 1
PCI_ENABLE 5
PCI_INTERRUPT_ENABLE 31
PCI_INTERRUPT_ENABLE 8
PCI_LTR_SCALE_SHIFT 10
PCI_MASTER_ABORT_RECEIVED_INTERRUPT_ENABLE 20
PCI_NUM_RESOURCES 17
PCI_RETRY_ABORT_INTERRUPT_ENABLE 17
PCI_TARGET_ABORT_RECEIVED_INTERRUPT_ENABLE 19
PCM_MULTICH 4
PCM_SPDIF 3
PCS_FW_HEARTB_WIDTH 8
PCS_UC_STATUS_LBN 0
PCS_UC_STATUS_WIDTH 8
PCTL_LINK_OFFS 16
PCXHR_DSP_RESET_GPO_OFFSET 5
PCXHR_FIRMWARE_DSP_EPRM_INDEX 2
PCXHR_FIRMWARE_DSP_MAIN_INDEX 4
PCXHR_FIRMWARE_XLX_COM_INDEX 1
PCXHR_PIPE_STATE_CAPTURE_OFFSET 12
PCXHR_SELMIC_PREAMPLI_OFFSET 2
PD_BITS 4
PDC_DIMM_BAR 4
PDC_MMIO_BAR 3
PDC_MMIO_BAR 5
PEER_ABORT 12
PEER_CLOSE 13
PERSISTENT_GNT_ACTIVE 0
PERSISTENT_GNT_WAS_ACTIVE 1
PFIT_HORIZ_SCALE_SHIFT 4
PFIT_PIPE_SHIFT 29
PFIT_VERT_SCALE_SHIFT 20
PF_VECTOR 14
PG_active 6
PG_CLEAN 2
PG_COMMIT_TO_DS 5
PG_dirty 4
PG_error 1
PG_LEVEL_1G 3
PG_LEVEL_2M 2
PG_locked 0
PG_lru 5
PG_MAPPED 1
PG_mlocked 21
PG_private 11
PG_reclaim 18
PG_referenced 2
PG_reserved 10
PG_swapbacked 19
PG_uptodate 3
PG_writeback 13
PHYAD_SHIFT 8
PHY_CTX_SHIFT 6
PHYID1_OUI_SHFT 6
PHY_TXC1_MODE_SHIFT 3
PHY_TXC_ANT_SHIFT 6
PIC_CASCADE_IR 2
PI_CONS_V_XMT_INDEX 16
```



```
PI_IO_CMP_V_SLOT 4
PIIX_SIDPR_BAR 5
PIPE_A 0
PIPE_B 1
PIPE_C 2
PI_RCV_DESCR_V_SEG_LEN 23
PI_XMT_DESCR_V_SEG_LEN 16
PKT_CLASSIFICATION_USER_PRIORITY_VALID 0
PKT_CLASSIFICATION_VLANID_VALID 1
PKTSShift 11
PLAYBACK_URB_COMPLETED 7
PLLCLK_IN_SHIFT 4
PLLD_LSB_SHIFT 2
PLLD_MSB_SHIFT 0
PLLJ_SHIFT 2
PLL_LOAD_PULSE_PHASE_SHIFT 9
PLLQ_SHIFT 3
PLL_REF_SDVO_HDMI_MULTIPLIER_SHIFT 9
PLL_REGM_F_SHIFT 0
PLL_REGM_SHIFT 9
PLL_REGN_SHIFT 1
PLLR_SHIFT 0
PLL_SD_SHIFT 9
PLL_SELFREQDCO_SHIFT 1
PM8607_IRQ_CC 5
PM8607_IRQ_ONKEY 0
PM8607_IRQ_RTC 4
PMA_PMD_EXT_SSR_LBN 15
PMA_PMD_FTX_STATIC_LBN 13
PMA_PMD_LNPGA_POWERDOWN_LBN 8
PMA_PMD_RXIN_SEL_LBN 6
PMU_XTALFREQ_REG_MEASURE_SHIFT 31
PORT_BASED_PRIORITY_SHIFT 3
PORT_DEVSLP_DETO_OFFSET 2
PORT_DEVSLP_DITO_OFFSET 15
PORT_DEVSLP_MDAT_OFFSET 10
PORT_FBS_DEV_OFFSET 8
PORT_IRQ_RAW_SHIFT 16
PPOD_IDX_SHIFT 6
PPOD_SIZE_SHIFT 6
PP_REFERENCE_DIVIDER_SHIFT 8
PPR_STATUS_SHIFT 12
PRB_MX_ADDR_MOD_SEL_SHIFT 9
PRISM2_INFO_PENDING_LINKSTATUS 0
PRISM2_INFO_PENDING_SCANRESULTS 1
PRIVATE_DTMF_TONE 5
PRIVATE_ECHO_CANCELLER 0
PRIVATE_FAX_NONSTANDARD 8
PRIVATE_FAX_PAPER_FORMATS 6
PRIVATE_FAX_SUB_SEP_PWD 3
PRIVATE_PIAFS 29
PRIVATE_RTP 1
PRIVATE_T38 2
PRIVATE_V18 4
PRIVATE_VOWN 7
PROFILE_GRPSHIFT 3
PSS_SHIFT 12
PSZ_CIF 4
PSZ_QCIF 2
PSZ_QSIF 1
PSZ_SIF 3
PSZ_SQCIF 0
PSZ_VGA 5
PTE_FILE_MAX_BITS 46
PT_OPT_FLAG_SHIFT 3
PTP_STNSUR_ADDSUB_SHIFT 31
```

```
PVR2_CVAL_INPUT_COMPOSITE 2
PVR2_CVAL_INPUT_DTV 1
PVR2_CVAL_INPUT_RADIO 4
PVR2_CVAL_INPUT_SVIDEO 3
PVR2_CVAL_INPUT_TV 0
PWIDX_ADC 0
PWIDX_CLFE 2
PWIDX_FRONT 1
PWIDX_MIC 4
PWIDX_SURR 3
PWMF_REQUESTED 1
PWM_NCHAN 4
QDF_CHANGE 1
QDF_LOCKED 2
QDF_REFRESH 3
__QDISC_STATE_DEACTIVATED 1
QFQ_MAX_WSHIFT 10
QFQ_MTU_SHIFT 16
QIB_7220_RcvCtrl_RcvQPMapEnable_LSB 38
QIB_7220_SendDmaLenGen_Generation_MSB 18
QIB_7322_SendDmaLenGen_0_Generation_MSB 18
QIB_AETH_CREDIT_SHIFT 24
QIB_CTXT_MASTER_UNINIT 4
QIB_CTXT_WAITING_RCV 2
QIB_CTXT_WAITING_URG 5
QIB_EEPROM_WEN_NUM 14
_QIB_EVENT_DISARM_BUFS_BIT 0
QIB_R_REWIND_SGE 1
QIB_R_WRID_VALID 0
QIF_DQBLKSIZE_BITS 10
QL_ADAPTER_UP 0
QL_ADAPTER_UP 7
QL_ALLMULTI 6
QL_ALLOC_BUFQS_DONE 11
QL_ALLOC_REQ_RSP_Q_DONE 10
QL_ALLOC_SMALL_BUF_DONE 12
QL_ASIC_RECOVERY 14
QLC_83XX_IDC_COMP_AEN 3
QLC_83XX_MBX_READY 2
QLC_83XX_MODULE_LOADED 1
QL_CAM_RT_SET 8
QLC_BC_CFREE 1
QLC_BC_FLR 2
QLC_BC_MSG 0
QLC_BC_VF_FLR 4
QLC_BC_VF_SOFT_FLR 5
QLC_BC_VF_STATE 3
__QLCNIC_AER 5
__QLCNIC_DEV_UP 1
__QLCNIC_DIAG_RES_ALLOC 6
__QLCNIC_FW_ATTACHED 0
__QLCNIC_RESETTING 2
__QLCNIC_SRIOV_CAPABLE 11
__QLCNIC_SRIOV_ENABLE 10
__QLCNIC_START_FW 4
QL_DMA64 4
QL_EEH_FATAL 12
QL_FRC_COREDUMP 11
QL_LB_LINK_UP 10
QL_LINK_MASTER 6
QL_LINK_OPTICAL 13
QL_MSI_ENABLED 14
QL_MSI_ENABLED 2
QL_MSIX_ENABLED 3
QLOGIC_IB_HWE_PCIEMEMPARITYERR_SHIFT 0
QLOGIC_IB_IBCC_LINKCMD_SHIFT 18
```



```
QLOGIC_IB_IBCC_LINKINITCMD_SHIFT 16
QLOGIC_IB_I_RCVAVAIL_SHIFT 0
QLOGIC_IB_I_RCVAVAIL_SHIFT 12
QLOGIC_IB_I_RCVURG_SHIFT 0
QLOGIC_IB_I_RCVURG_SHIFT 32
QLOGIC_IB_R_INTRAVAIL_SHIFT 16
QLOGIC_IB_R_TAILUPD_SHIFT 31
QLOGIC_IB_SENDPIOAVAIL_BUSY_SHIFT 1
QL_PORT_CFG 7
QL_PROMISCUOUS 5
QL_RESET_ACTIVE 2
QL_RESET_DONE 1
QL_RESET_PER_SCSI 4
QL_RESET_START 3
QL_SELFTEST 9
QP_REFERENCED 5
QSFP_GPIO_PORT2_SHIFT 5
QS_MMIO_BAR 4
QUEUE_FLAG_BIDI 7
QUEUE_FLAG_DISCARD 14
QUEUE_FLAG_NONROT 12
QUEUE_PAUSED 1
QUN 4
QUOTABLOCK_BITS 10
R10BIO_Degraded 4
R10BIO_IsRecover 2
R10BIO_IsReshape 3
R10BIO_IsSync 1
R10BIO_MadeGood 6
R10BIO_Previous 8
R10BIO_ReadError 5
R10BIO_Uptodate 0
R10BIO_WriteError 7
R1BIO_BehindIO 3
R1BIO_Degraded 2
R1BIO_IsSync 1
R1BIO_MadeGood 7
R1BIO_ReadError 4
R1BIO_Uptodate 0
R1BIO_WriteError 8
R300_MC_DISP0R_INIT_LAT_SHIFT 8
R300_MC_DISP1R_INIT_LAT_SHIFT 12
R300_MSBD0_X_SHIFT 28
R300_MSBD0_Y_SHIFT 24
R300_MSBD1_SHIFT 24
R300_MS_X0_SHIFT 0
R300_MS_X1_SHIFT 8
R300_MS_X2_SHIFT 16
R300_MS_X3_SHIFT 0
R300_MS_X4_SHIFT 8
R300_MS_X5_SHIFT 16
R300_MS_Y0_SHIFT 4
R300_MS_Y1_SHIFT 12
R300_MS_Y2_SHIFT 20
R300_MS_Y3_SHIFT 4
R300_MS_Y4_SHIFT 12
R300_MS_Y5_SHIFT 20
R592_TPC_EXEC_LEN_SHIFT 16
R592_TPC_EXEC_TPC_SHIFT 28
R5_Discard 23
R5_DOUBLE_LOCKED 2
R5_Expanded 11
R5_Insync 4
R5_LOCKED 1
R5_MadeGood 18
R5_MadeGoodRepl 20
```

```
R5_NeedReplace 21
R5_Overlap 7
R5_OVERWRITE 3
R5_ReadError 9
R5_ReadNoMerge 8
R5_ReadRepl 19
R5_ReWrite 10
R5_SyncIO 16
R5_UPTODATE 0
R5_Wantcompute 12
R5_Wantdrain 14
R5_Wantfill 13
R5_WantFUA 15
R5_Wantread 5
R5_WantReplace 22
R5_Wantwrite 6
R5_WriteError 17
RADEON_ACTIVE_HILO_LAT_SHIFT 13
RADEON_BLANK_LEVEL_SHIFT 8
RADEON_CRT1_CRTC_SHIFT 9
RADEON_CRT2_CRTC_SHIFT 12
RADEON_CRTC2_PIX_WIDTH_SHIFT 8
RADEON_CRTC_H_DISP_SHIFT 16
RADEON_CRTC_H_SYNC_STRT_CHAR_SHIFT 3
RADEON_CRTC_H_TOTAL_SHIFT 0
RADEON_CRTC_V_DISP_SHIFT 16
RADEON_CRTC_V_SYNC_STRT_SHIFT 0
RADEON_CRTC_V_TOTAL_SHIFT 0
RADEON_CY_FILT_BLEND_SHIFT 28
RADEON_DAC_FORCE_DATA_SHIFT 8
RADEON_DFP1_CRTC_SHIFT 11
RADEON_DFP2_CRTC_SHIFT 14
RADEON_DPM_AUTO_THROTTLE_SRC_EXTERNAL 1
RADEON_DPM_AUTO_THROTTLE_SRC_THERMAL 0
RADEON_FIFORAM_FFMACRO_READ_MARGIN_SHIFT 20
RADEON_GPU_PAGE_SHIFT 12
RADEON_GRPH_START_REQ_SHIFT 0
RADEON_GRPH_STOP_REQ_SHIFT 8
RADEON_H_INC_SHIFT 0
RADEON_I2C_PRESCALE_SHIFT 16
RADEON_LCD1_CRTC_SHIFT 8
RADEON_LVDS_BL_MOD_LEVEL_SHIFT 8
RADEON_LVDS_PWRSEQ_DELAY1_SHIFT 16
RADEON_LVDS_PWRSEQ_DELAY2_SHIFT 20
RADEON_NUM_VERTICES_SHIFT 16
RADEON_PCIE_LC_LINK_WIDTH_RD_SHIFT 4
RADEON_PCIE_LC_LINK_WIDTH_SHIFT 0
RADEON_SET_UP_LEVEL_SHIFT 16
RADEON_SPLL_FB_DIV_SHIFT 16
RADEON_SPLL_PVG_SHIFT 11
RADEON_TV1_CRTC_SHIFT 10
RADEON_TV_DAC_BGADJ_SHIFT 16
RADEON_TV_DAC_DACADJ_SHIFT 20
RADEON_TV_FORCE_DAC_DATA_SHIFT 16
RADEON_TV_M0HI_SHIFT 18
RADEON_TV_N0HI_SHIFT 21
RADEON_TV_N0LO_SHIFT 8
RADEON_TVPCP_SHIFT 8
RADEON_TVPDC_SHIFT 14
RADEON_TV_P_SHIFT 24
RADEON_TVPVG_SHIFT 11
RADEON_TV_V_BURST_LEVEL_SHIFT 16
RADEON_UV_OUTPUT_POST_SCALE_SHIFT 24
RADEON_UVRAM_READ_MARGIN_SHIFT 16
RADEON_Y_DEL_W_SIG_SHIFT 26
RADIO_2055_NBRSSI_VCM_I_SHIFT 0
```



```
RADIO_2055_NBRSSI_VCM_Q_SHIFT 0
RADIO_2055_WBRSSI_VCM_IQ_SHIFT 2
RADIO_2056_RSSI_VCM_SHIFT 2
RAMROD_CONT 6
RAMROD_DRV_CLR_ONLY 3
RAMROD_EXEC 5
RAMROD_RX 1
RAMROD_TX 0
RATELIMIT_CALC_SHIFT 10
RATE_MCS_ANT_POS 14
RATIO_SCALE_LOG 8
RB_BUFFERS_DISABLED_BIT 1
RBD_DEV_FLAG_EXISTS 0
RBIO_CACHE_READY_BIT 3
RBIO_RMW_LOCKED_BIT 1
RBP_IDX_OFFSET 6
RBR_CFIG_B_BLKSIZE_SHIFT 24
RBR_CFIG_B_BUFSZ0_SHIFT 0
RBR_CFIG_B_BUFSZ1_SHIFT 8
RBR_CFIG_B_BUFSZ2_SHIFT 16
RBSShift 0
RC_MODEL_TOTAL_BITS 11
RC_PID_ARITH_SHIFT 8
RCR_ENTRY_PKT_BUF_ADDR_SHIFT 6
RCR_FIFO_OFFSET 13
RC_SHIFT_BITS 8
RC_TOP_BITS 24
RCV_LAZY_FC_SHIFT 24
RDC_RED_PARA_THRE_SHIFT 4
RDC_RED_PARA_THRE_SYN_SHIFT 20
RDC_RED_PARA_WIN_SHIFT 0
RDC_RED_PARA_WIN_SYN_SHIFT 16
RDMACTXT_F_FAST_UNREG 1
RDMACTXT_F_LAST_CTXT 2
RDMAXPRT_CONN_PENDING 3
RDMAXPRT_RQ_PENDING 1
RDMAXPRT_SQ_PENDING 2
RDS_LL_SEND_FULL 0
RDS_MSG_ACK_REQUIRED 4
RDS_MSG_HAS_ACK_SEQ 3
RDS_MSG_MAPPED 6
RDS_MSG_ON_CONN 2
RDS_MSG_ON_SOCK 1
RDS_MSG_PAGEVEC 7
RDS_MSG_RETRANSMITTED 5
RDS_RECONNECT_PENDING 1
READ_LOC_AMP_ASSOC 1
READ_LOC_AMP_ASSOC_FINAL 2
READ_LOC_AMP_INFO 0
RECOVERY_CLEANUP 8
RED_LED 1
RED_LED 2
REDUCE_REPORTING 3
REGISTER_FC4_NEEDED 9
REGISTER_FDMI_NEEDED 12
reg_ofdm_rst_pos 1
REISERFS_3_5 0
REISERFS_3_6 1
REISERFS_ATTRS 14
REISERFS_BARRIER_FLUSH 19
REISERFS_BARRIER_NONE 18
REISERFS_CONVERT 3
REISERFS_DATA_LOG 8
REISERFS_DATA_ORDERED 9
REISERFS_DATA_WRITEBACK 10
REISERFS_ERROR_CONTINUE 22
```

```
3163 REISERFS_ERROR_PANIC 20
3164 REISERFS_ERROR_RO 21
3165 REISERFS_EXPOSE_PRIVROOT 17
3166 REISERFS_GRPQUOTA 24
3167 REISERFS_HASHED_RELOCATION 13
3168 REISERFS_LARGETAIL 0
3169 REISERFS_NO_BORDER 11
3170 REISERFS_NO_UNHASHED_RELOCATION 12
3171 REISERFS_OLD_FORMAT 2
3172 REISERFS_OPT_ALLOWEMPTY 31
3173 REISERFS_POSIXACL 16
3174 REISERFS_SMALLTAIL 1
3175 REISERFS_TEST4 28
3176 REISERFS_UNSUPPORTED_OPT 29
3177 REISERFS_USRQUOTA 23
3178 REISERFS_XATTRS_USER 15
3179 RELEASE_RESOURCES 2
3180 REL_HWHEEL 6
3181 RELOGIN_NEEDED 8
3182 REL_WHEEL 8
3183 REL_X 0
3184 REL_Y 1
3185 REMOTE_WAKEUP_SUPPORT 5
3186 Replacement 10
3187 REPLAYONLY 2
3188 RESET_MARKER_NEEDED 0
3189 RESIZE_PENDING 16
3190 RESOLVE_CONFLICTS 1
3191 RESUME_INTERRUPT 1
3192 RESUME_INTERRUPT 5
3193 RESYNC_AFTER_NEG 15
3194 RETRY_LIMIT_LONG_SHIFT 0
3195 RETRY_LIMIT_SHORT_SHIFT 8
3196 RFCOMM_AUTH_ACCEPT 6
3197 RFCOMM_AUTH_PENDING 5
3198 RFCOMM_AUTH_REJECT 7
3199 RFCOMM_DEFER_SETUP 8
3200 RFCOMM_ENC_DROP 9
3201 RFCOMM_HANGUP_NOW 2
3202 RFCOMM_MSC_PENDING 3
3203 RFCOMM_RELEASE_ONHUP 1
3204 RFCOMM_REUSE_DLC 0
3205 RFCOMM_SEC_PENDING 4
3206 RFCOMM_TIMED_OUT 2
3207 RFCOMM_TTY_ATTACHED 3
3208 RFCOMM_TX_THROTTLED 1
3209 RH_OFS 0
3210 RISC_RESET_STATUS_SHIFT 20
3211 RISC_SET_STATUS_SHIFT 16
3212 ROAM_TBL_PEND 6
3213 ROOT_PORT_RESET_INTERRUPT 4
3214 ROOT_PORT_RESET_INTERRUPT 6
3215 ROOT_PORT_RESET_INTERRUPT_ENABLE 4
3216 ROOT_PORT_RESET_INTERRUPT_ENABLE 6
3217 R_OPCODE_LSB 3
3218 RPCAUTH_CRED_HASHED 2
3219 RPCAUTH_CRED_NEGATIVE 3
3220 RPCAUTH_CRED_NEW 0
3221 RPCAUTH_CRED_UPTODATE 1
3222 RPC_BC_PA_IN_USE 1
3223 RPC_TASK_ACTIVE 2
3224 Rpending 2
3225 RSCN_UPDATE 7
3226 RSCONFIG_MAX_DMA_SIZE_SHIFT 24
3227 RSCONFIG_STREAM_NUM_SHIFT 16
3228 RS_OFS 16
```



```
3229 RSPEC_BW_SHIFT 8
3230 RSPEC_STC_SHIFT 20
3231 RSPEC_STF_SHIFT 11
3232 RST_OFS 8
3233 RT5640_DMIC_CLK_SFT 5
3234 RT5640_I2S_BCLK_MS1_SFT 15
3235 RT5640_I2S_BCLK_MS2_SFT 11
3236 RT5640_I2S_PD1_SFT 12
3237 RT5640_I2S_PD2_SFT 8
3238 RT5640_PLL_M_BP_SFT 11
3239 RT5640_PLL_M_SFT 12
3240 RT5640_PLL_N_SFT 7
3241 RTC_RBUDR_SHIFT 4
3242 RTC_UDR_SHIFT 0
3243 R_TDI_LSB 2
3244 RT_IDX_IDX_SHIFT 8
3245 RT_IDX_TYPE_SHIFT 16
3246 RTL8150_HW_CRC 0
3247 RTL8150_UNPLUG 2
3248 RTL8152_SET_RX_MODE 2
3249 RTL8152_UNPLUG 0
3250 RTL_FLAG_TASK_ENABLED 0
3251 RTL_FLAG_TASK_RESET_PENDING 2
3252 RTL_STATUS_INTERFACE_START 0
3253 RTSCTS_SH_CTS_MOD_TYPE 24
3254 RTSCTS_SH_CTS_PMB_TYPE 25
3255 RTSCTS_SH_CTS_RATE 16
3256 RTSCTS_SH_RTS_MOD_TYPE 8
3257 RTSCTS_SH_RTS_PMB_TYPE 9
3258 RTSCTS_SH_RTS_RATE 0
3259 RTS_LEVEL_SHIFT_BITS 2
3260 RXBSIDH_SHIFT 3
3261 RxBufferLenShift 16
3262 RxBurstSizeShift 0
3263 RxComplThreshShift 0
3264 RXD_FLAGS_SHIFT 0
3265 RXD_LEN_SHIFT 0
3266 RX_DMA_CTL_STAT_PKTREAD_SHIFT 0
3267 RX_DMA_CTL_STAT_PTRREAD_SHIFT 16
3268 RXD_OPAQUE_INDEX_SHIFT 0
3269 RxDRNT_shift 1
3270 RxEarlyIntThreshShift 12
3271 RX_FLAG_STBC_SHIFT 26
3272 RX_FLOW_ON_BIT 2
3273 RxHighPrioThreshShift 8
3274 RX_LOG_PAGE_VLD_FUNC_SHIFT 2
3275 RxMinDescrThreshShift 0
3276 RxMXDMA_shift 20
3277 RXQ_CTRL_RFD_BURST_NUM_SHIFT 0
3278 RXQ_CTRL_RFD_PREF_MIN_IPG_SHIFT 16
3279 RXQ_CTRL_RRD_BURST_THRESH_SHIFT 8
3280 RXQ_JMBO_LKAH_SHIFT 11
3281 RXQ_JMBOSZ_TH_SHIFT 0
3282 RXQ_RFD_BURST_NUM_SHIFT 20
3283 RXQ_RRD_PAUSE_TH_HI_SHIFT 0
3284 RXQ_RRD_PAUSE_TH_LO_SHIFT 16
3285 RXQ_RRD_TIMER_SHIFT 16
3286 RXQ_RXF_PAUSE_TH_HI_SHIFT 0
3287 RXQ_RXF_PAUSE_TH_HI_SHIFT 16
3288 RXQ_RXF_PAUSE_TH_LO_SHIFT 0
3289 RXQ_RXF_PAUSE_TH_LO_SHIFT 16
3290 RXRPC_CALL_ABORT 7
3291 RXRPC_CALL_ACK_FINAL 4
3292 RXRPC_CALL_CONN_ABORT 8
3293 RXRPC_CALL_DRAIN_RX_OOS 11
3294 RXRPC_CALL_LIFE_TIMER 12
```

```
RXRPC_CALL_POST_ACCEPT 15
RXRPC_CALL_RCVD_ABORT 2
RXRPC_CALL_RCVD_ACKALL 0
RXRPC_CALL_RCVD_BUSY 1
RXRPC_CALL_RCVD_ERROR 3
RXRPC_CALL_RCVD_LAST 2
RXRPC_CALL_RESEND 10
RXRPC_CALL_RUN_RTIMER 3
RXRPC_CALL_TERMINAL_MSG 1
RXRPC_CALL_TX_SOFT_ACK 4
RXRPC_CONN_CHALLENGE 0
RXRPC_SOCK_EXCLUSIVE_CONN 1
RX_URB_FAIL 1
RX_URB_FAIL 3
__S2IO_STATE_CARD_UP 1
S5M8767_ENCTRL_SHIFT 6
SA2400_REG4_FIRDAC_SHIFT 7
S_ALIVE 6
SAR_TBD_VCI_SHIFT 4
SAR_TBD_VPI_SHIFT 20
SAS_DEV_EH_PENDING 3
SAS_DEV_FOUND 1
SAS_DEV_GONE 0
SAS_HA_ATA_EH_ACTIVE 2
SAS_HA_DRAINING 1
SAS_HA_FROZEN 3
SAS_HA_REGISTERED 0
SATA_PMP_CTRL_PORT 15
SAVAGE_BD_BPP_SHIFT 16
SAVAGE_BD_TILE_SHIFT 24
SB_M_DIVIDER_SHIFT 24
SB_N_CB_TUNE_SHIFT 24
SB_N_DIVIDER_SHIFT 26
SB_N_VCO_SEL_SHIFT 30
SBSDIO_FUNC1_SLEEPCSR_KSO_SHIFT 0
SBSDIO_FUNC1_WCTRL_HTWAIT_SHIFT 1
SCAN_ABORTED 4
SCAN_COMPLETED 3
S_CHANNEL_SWITCH_PENDING 18
SCHED_SCANNING 12
SC_OP_ANI_RUN 2
SC_OP_BEACONS 1
SC_OP_HW_RESET 4
SC_OP_INVALID 0
SC_OP_PRIM_STA_VIF 3
SC_OP_SCANNING 5
SC_RA_RAM_BE_OPT_ENA_CP_OPT 1
SCR_PENDING 21
SCU_COMPLETION_TL_STATUS_SHIFT 22
SCU_CONTEXT_COMMAND_LOGICAL_PORT_SHIFT 12
SCU_CONTEXT_COMMAND_PROTOCOL_ENGINE_GROUP_SHIFT 16
SCU_EVENT_SPECIFIC_CODE_SHIFT 18
SCU_RAM_AGC_KI_IF__B 8
SCU_RAM_AGC_KI_RED_IAGC_RED__B 4
SCU_RAM_AGC_KI_RED_RAGC_RED__B 2
SCU_RAM_AGC_KI_RF__B 4
SD1_DVM_EN 6
SD1_DVM_SHIFT 5
SDATA_STATE_OFFCHANNEL 1
SDATA_STATE_OFFCHANNEL_BEACON_STOPPED 2
SDATA_STATE_RUNNING 0
SDCLK_DELAY_SHIFT 10
SDCLK_DELAY_SHIFT 9
SDCLK_SEL_SHIFT 8
SD_CTL_STREAM_TAG_SHIFT 20
SDEV_EVT_MEDIA_CHANGE 1
```



```
SDF_JOURNAL_CHECKED 0
SDF_JOURNAL_LIVE 1
SDF_NOBARRIERS 3
SDF_NOJOURNALID 6
SDF_NORECOVERY 4
SDF_RORECOVERY 7
SDF_SKIP_DLM_UNLOCK 8
SDHCI_CLOCK_BASE_SHIFT 8
SDHCI_DIVIDER_HI_SHIFT 6
SDHCI_DIVIDER_SHIFT 8
SDHCI_MAX_CURRENT_180_SHIFT 16
SDHCI_MAX_CURRENT_300_SHIFT 8
SDHCI_MAX_CURRENT_330_SHIFT 0
SDHCI_TIMEOUT_CLK_SHIFT 0
SDIO_DRIVE_DTSx_SHIFT 4
SDIO_FUNC_1 1
SDIO_FUNC_2 2
SDMA_DESC_COUNT_LSB 16
SDMA_DESC_GEN_LSB 30
SDM_OP_GEN_AGG_VECT_IDX_VALID_SHIFT 16
SDPCM_CHANNEL_SHIFT 8
SDPCM_DOFFSET_SHIFT 24
SDVO_MULTIPLIER_SHIFT_HIRES 4
SDVO_PORT_MULTIPLY_SHIFT 23
SECTION_NID_SHIFT 2
SECTOR_BITS 9
SECTOR_SHIFT 9
SELF_POWERED_STATUS 0
SELF_POWERED_USB_DEVICE 6
SEM_FLASH_SHIFT 8
SEM_ICB_SHIFT 4
SEM_MAC_ADDR_SHIFT 6
SEM_PROBE_SHIFT 10
SEM_PROC_REG_SHIFT 14
SEM_RT_IDX_SHIFT 12
SEM_XGMAC0_SHIFT 0
SEM_XGMAC1_SHIFT 2
SEND_PING 2
SEP_FASTCALL_WRITE_DONE_OFFSET 0
SEP_LEGACY_MMAP_DONE_OFFSET 1
SEP_LEGACY_POLL_DONE_OFFSET 3
SEP_LEGACY_SENDMSG_DONE_OFFSET 2
SEQNUM_SHIFT 4
SERDES_CHANS 4
SERPORT_ACTIVE 2
SERPORT_DEAD 3
SET_ADDRESS 13
SET_ENDPOINT_HALT 8
SET_NAK_OUT_PACKETS 15
SET_NAK_OUT_PACKETS_MODE 10
SET_TEST_MODE 16
SETUP_PACKET_INTERRUPT 5
SETUP_PACKET_INTERRUPT 7
SETUP_PACKET_INTERRUPT_ENABLE 5
SETUP_PACKET_INTERRUPT_ENABLE 7
SE_VAP_CNTL__VF_MAX_VTX_NUM__SHIFT 18
S_EXIT_PENDING 10
SFQ_ALLOT_SHIFT 3
S_FW_ERROR 17
S_GEO_CONFIGURED 9
SGTL5000_ANA_GND_SHIFT 4
SGTL5000_BIAS_R_SHIFT 8
SGTL5000_DAC_SEL_SHIFT 4
SGTL5000_DAC_VOL_LEFT_SHIFT 0
SGTL5000_DAC_VOL_RIGHT_SHIFT 8
SGTL5000_I2S_DLEN_SHIFT 4
```

```
SGTL5000_I2S_SCLKFREQ_SHIFT 8
SGTL5000_LINE_OUT_CURRENT_SHIFT 8
SGTL5000_LINE_OUT_GND_SHIFT 0
SGTL5000_MCLK_FREQ_SHIFT 0
SGTL5000_PLL_FRAC_DIV_SHIFT 0
SGTL5000_PLL_INT_DIV_SHIFT 11
SGTL5000_RATE_MODE_SHIFT 4
SGTL5000_SYS_FS_SHIFT 2
SGTL5000_VDDC_MAN_ASSN_SHIFT 6
SHADOW_REG_SHIFT 20
SHARED_HW_CFG_LED_MODE_SHIFT 16
S_HCMD_ACTIVE 0
SHIFT 11
SHIFT_2 2
SHIFT 8
SHMEM_EEE_ADV_STATUS_SHIFT 20
SHMEM_EEE_LP_ADV_STATUS_SHIFT 24
SHMEM_EEE_SUPPORTED_SHIFT 16
SHORT_PACKET_OUT_DONE_INTERRUPT 6
SHORT_PACKET_TRANSFERRED_INTERRUPT 4
SHORT_PACKET_TRANSFERRED_INTERRUPT 5
SHORT_PACKET_TRANSFERRED_INTERRUPT_ENABLE 5
SHUTDOWN_MERGE 1
SI476X_DIGITAL_IO_SAMPLE_SIZE_SHIFT 8
SI476X_DIGITAL_IO_SLOT_SIZE_SHIFT 11
SI5351_OUTPUT_CLK_DIV_SHIFT 4
SIDL_EXIDE_SHIFT 3
SIDL_SID_SHIFT 5
SIGNAL_ASENDER 3
SIL164_CONTROL1_DESKEW_INCR_SHIFT 5
SIL164_CONTROL2_FILTER_SETTING_SHIFT 1
SIL164_DUALLINK_SKEW_SHIFT 5
SIL24_HOST_BAR 0
SIL24_PORT_BAR 2
SIL680_MMIO_BAR 5
SIL_MMIO_BAR 5
S_INIT 5
SIO_HI_RA_RAM_PAR_3_CFG_DBL_SCL__B 7
SIO_PDR_MCLK_CFG_DRIVE__B 3
SIO_PDR_MD0_CFG_DRIVE__B 3
SIS_MM_ALIGN_SHIFT 0
SIS_SCR_PCI_BAR 5
SKB_DATAREF_SHIFT 16
SK_F_HW_ERR 8
SK_F_IPV6 5
SK_F_OFFLD_COMPLETE 1
SK_F_PG_OFFLD_COMPLETE 3
SLF_ERROR 1
SLF_ERROR 2
SLF_ESCAPE 1
SLF_INUSE 0
SLF_KEEPTEST 3
SLF_OUTWAIT 4
SLICE_BITS 4
SMB_DATA_VERSION_SHIFT 16
SMBINTF_HWADDR_SHIFT 8
SMBINTF_HWDATW_SHIFT 16
SMC_NISLANDS_SPLL_DIV_TABLE_CLKS_SHIFT 20
SMC_NISLANDS_SPLL_DIV_TABLE_CLKV_SHIFT 0
SMC_NISLANDS_SPLL_DIV_TABLE_FBDIV_SHIFT 0
SMC_NISLANDS_SPLL_DIV_TABLE_PDIV_SHIFT 25
SMC_SISLANDS_SCALE_R 12
SMC_SISLANDS_SPLL_DIV_TABLE_CLKS_SHIFT 20
SMC_SISLANDS_SPLL_DIV_TABLE_CLKV_SHIFT 0
SMC_SISLANDS_SPLL_DIV_TABLE_FBDIV_SHIFT 0
SMC_SISLANDS_SPLL_DIV_TABLE_PDIV_SHIFT 25
```



```
SMI_DATA_SHIFT 16
SMP_FLAG_CFM_PENDING 2
SMP_FLAG_MITM_AUTH 3
SMP_FLAG_TK_VALID 1
SMPL_EN_SHIFT 7
SMPLT_SHIFT 2
SNDRV_PCM_HW_PARAM_RATE 11
SNDRV_PCM_HW_PARAM_SAMPLE_BITS 8
SNDRV_PCM_STREAM_CAPTURE 1
SNDRV_PCM_STREAM_PLAYBACK 0
SNDRV_TIMER_EVENT_CONTINUE 4
SNDRV_TIMER_EVENT_MCONTINUE 14
SNDRV_TIMER_EVENT_MPAUSE 15
SNDRV_TIMER_EVENT_MRESUME 18
SNDRV_TIMER_EVENT_MSTART 12
SNDRV_TIMER_EVENT_MSTOP 13
SNDRV_TIMER_EVENT_MSUSPEND 17
SNDRV_TIMER_EVENT_PAUSE 5
SNDRV_TIMER_EVENT_RESOLUTION 0
SNDRV_TIMER_EVENT_RESUME 8
SNDRV_TIMER_EVENT_START 2
SNDRV_TIMER_EVENT_STOP 3
SNDRV_TIMER_EVENT_SUSPEND 7
SNDRV_TIMER_EVENT_TICK 1
SOCK_ASYNC_NOSPACE 0
SOCK_ASYNC_WAITDATA 1
SOCK_EXTERNALLY_ALLOCATED 5
SOCK_NOSPACE 2
SOCK_PASSCRED 3
SOCK_PASSSEC 4
SOCK_TIMESTAMPING_RX_SOFTWARE 19
SOF_DOWN_INTERRUPT 14
SOF_INTERRUPT 0
SOF_INTERRUPT 7
SOFTACK_MSHIFT 15
SOFTACK_PSHIFT 16
SPCR_RFIC_FIELD 20
SPCR_TFIC_FIELD 16
SPE_HDR_CONN_TYPE_SHIFT 0
SPE_HDR_FUNCTION_ID_SHIFT 8
SPI_FLASH_CTRL_CLK_HI_SHIFT 22
SPI_FLASH_CTRL_CLK_LO_SHIFT 20
SPI_FLASH_CTRL_CS_HI_SHIFT 16
SPI_FLASH_CTRL_CS_HOLD_SHIFT 18
SPI_FLASH_CTRL_CS_SETUP_SHIFT 24
SPI_FLASH_CTRL_INS_SHIFT 8
SPI_FRF_OFFSET 4
SPI_MODE_OFFSET 6
SPI_REG_SHIFT 9
SPI_SS_OFF 2
SPI_SS_ON 1
SPI_TERMINATE 3
SPI_TMOD_OFFSET 8
SPI_XCOMM_SETTINGS_LEN_OFFSET 10
SPI_XFER 0
S_PORT 1
S_POWER_PMI 16
S_READY 7
S_RFKILL 3
S_RXCOALESCESIZE 0
SSB_CHIPCO_PMU_CTL_ILP_DIV_SHIFT 16
SSB_CHIPCO_PMU_CTL_XTALFREQ_SHIFT 2
SSB_FLASH_WCNT_1_SHIFT 8
SSB_FLASH_WCNT_3_SHIFT 24
SSB_IMCFGLO_REQTO_SHIFT 4
SSB_PMU0_PLLCTL1_WILD_FMSK_SHIFT 8
```

```
SSB_PMU0_PLLCTL1_WILD_IMSK_SHIFT 28
SSB_PMU0_PLLCTL2_WILD_IMSKHI_SHIFT 0
SSB_PMU1_PLLCTL0_P1DIV_SHIFT 20
SSB_PMU1_PLLCTL0_P2DIV_SHIFT 24
SSB_PMU1_PLLCTL2_NDIVINT_SHIFT 20
SSB_PMU1_PLLCTL2_NDIVMODE_SHIFT 17
SSB_PMU1_PLLCTL3_NDIVFRAC_SHIFT 0
SSB_PMU1_PLLCTL5_CLKDRV_SHIFT 8
SSB_PMURES_4325_CBUCK_BURST 1
SSB_PMURES_4325_CLDO_CBUCK_BURST 3
SSB_PMURES_4325_LNLDO2_PU 10
SSB_PMURES_4328_BB_SWITCHER_PWM 1
SSB_PMURES_4328_EXT_SWITCHER_PWM 0
SSB_PMURES_4328_XTAL_EN 16
SSB_PROG_WCNT_1_SHIFT 8
SSB_PROG_WCNT_2_SHIFT 16
SSB_PROG_WCNT_3_SHIFT 24
S_SCAN_HW 15
S_SCANNING 13
SSM2518_CHAN_MAP_LEFT_SLOT_OFFSET 0
SSM2518_CHAN_MAP_RIGHT_SLOT_OFFSET 4
S_STATS 12
STA32X_CONFA_IR_SHIFT 3
STA32X_CONFA_MCS_SHIFT 0
STA32X_CONFF_OCFG_SHIFT 0
STA32X_CxCFG_OM_SHIFT 6
STA_KEY_FLG_KEYID_POS 8
STATE_SENT 10
STATIC_MAC_FID_SHIFT 22
STATIC_MAC_FWD_PORTS_SHIFT 16
STAT_RDA 7
STATS_UPDATE_PEND 8
STATUS_ALIVE 2
STATUS_CHANNEL_SWITCH_PENDING 11
STATUS_CT_KILL 1
STATUS_DEVICE_ENABLED 1
STATUS_EXIT_PENDING 5
STATUS_FW_ERROR 10
STATUS_FW_ERROR 5
STATUS_READY 3
STATUS_RFKILL 4
STATUS_RF_KILL_HW 0
STATUS_SCAN_COMPLETE 12
STATUS_SCAN_HW 9
STATUS_SCANNING 7
STATUS_SCAN_ROC_EXPIRED 14
STATUS_STATISTICS 6
STATUS_TPOWER_PMI 2
STB6100_K_PSD2_SHIFT 2
STB6100_VCO_ODIV_SHIFT 4
S_TEMPERATURE 8
STMPE_IRQ_TOUCH_DET 0
STOP_TX 1
STREAM_FMT_OFFSET 10
STREAM_INDEP_HP 1
STREAM_MULTI_OUT 0
ST_REG_IN_PROGRESS 2
ST_REG_PENDING 3
STRIPE_BIOFILL_RUN 14
STRIPE_BIT_DELAY 8
STRIPE_COMPUTE_RUN 15
STRIPE_DEGRADED 7
STRIPE_DELAYED 6
STRIPE_DISCARD 18
STRIPE_EXPANDING 9
STRIPE_EXPAND_READY 11
```



```
STRIPE_EXPAND_SOURCE 10
STRIPE_HANDLE 1
STRIPE_INSYNC 4
STRIPE_IO_STARTED 12
STRIPE_OP_BIODRAIN 3
STRIPE_OP_BIOFILL 0
STRIPE_OP_CHECK 5
STRIPE_OP_COMPUTE_BLK 1
STRIPE_OP_PREXOR 2
STRIPE_OP_RECONSTRUCT 4
STRIPE_SYNCING 3
STRIPE_SYNC_REQUESTED 2
ST_TX_WAKEUP 2
SUM_CHECK_P 0
SUM_CHECK_Q 1
SUNI_TPOP_APM_S_SHIFT 2
SUSPEND_IO 8
SUSPEND_REQUEST_CHANGE_INTERRUPT 2
SUSPEND_REQUEST_CHANGE_INTERRUPT 4
SUSPEND_REQUEST_CHANGE_INTERRUPT_ENABLE 2
SUSPEND_REQUEST_CHANGE_INTERRUPT_ENABLE 4
SUSPEND_REQUEST_INTERRUPT 3
S_VLANEXTRACTIONENABLE 12
SVM_EXITINFOSHIFT_TS_HAS_ERROR_CODE 44
SVM_EXITINFOSHIFT_TS_REASON_IRET 36
SVM_EXITINFOSHIFT_TS_REASON_JMP 38
SVM_IOIO_SIZE_SHIFT 4
SVM_SELECTOR_AVL_SHIFT 8
SVM_SELECTOR_DB_SHIFT 10
SVM_SELECTOR_DPL_SHIFT 5
SVM_SELECTOR_G_SHIFT 11
SVM_SELECTOR_L_SHIFT 9
SVM_SELECTOR_P_SHIFT 7
SVM_SELECTOR_S_SHIFT 4
SVM_VM_CR_SVM_DISABLE 4
SWDPIO__EXT_SHIFT 4
SW_LID 0
SW_TABLET_MODE 1
SYCC_CD_SHIFT 16
SYM_MEM_SHIFT 4
SYSFS_NS_TYPE_SHIFT 8
T3_CTRL_QP_SIZE_LOG2 8
TABLE_SEL_SHIFT 2
TA_READY_INPUT_ENABLE 6
TBSShift 0
TCAM_ASSOCDATA_OFFSET_SHIFT 2
TCAM_V4KEY0_CLASS_CODE_SHIFT 3
TCAM_V4KEY1_L2RDCNUM_SHIFT 59
TCAM_V4KEY2_PROTO_SHIFT 32
TCAM_V4KEY2_TOS_SHIFT 40
TCAM_V4KEY3_SADDR_SHIFT 32
TC_CBQ_MAXPRIO 8
TCE_READ_SHIFT 0
TCE_RPN_SHIFT 12
TCE_WRITE_SHIFT 1
TCODE_PHY_PACKET 16
TCON_LINK_IN_TREE 2
TCON_LINK_MASTER 0
TCON_LINK_PENDING 1
TCP_DELACK_TIMER_DEFERRED 4
TCP_METRIC_CWND 3
TCP_METRIC_REORDERING 4
TCP_METRIC_RTT 0
TCP_METRIC_RTTVAR 1
TCP_METRIC_SSTHRESH 2
TCP_MTU_REDUCED_DEFERRED 5
```

```
TCP_TSQ_DEFERRED 2
TCP_WRITE_TIMER_DEFERRED 3
TD_PIDEP_OFFSET 4
TD_TOKEN_DEVADDR_SHIFT 8
TEA5777_W_AM_FM_SHIFT 46
TEA5777_W_AM_LNA_SHIFT 30
TEA5777_W_AM_MWLW_SHIFT 31
TEA5777_W_AM_PEAK_SHIFT 25
TEA5777_W_AM_PLL_SHIFT 34
TEA5777_W_CHP0_SHIFT 18
TEA5777_W_FM_FORCEMONO_SHIFT 15
TEA5777_W_FM_FREF_SHIFT 30
TEA5777_W_FM_PLL_SHIFT 32
TEA5777_W_IFCE_SHIFT 29
TEA5777_W_IFW_SHIFT 28
TEA5777_W_INTEXT_SHIFT 24
TEA5777_W_SLEV_SHIFT 3
TESTMODE 3
TG3_PHY_MII_ADDR 1
THREAD_WAKEUP 0
TILES_IN_X_LSB_SHIFT 30
TILES_IN_X_MSB_SHIFT 24
TILES_IN_X_SHIFT 4
TIMEDOUT 11
TIMEOUT 0
TIMEOUT 21
TIPC_CRITICAL_IMPORTANCE 3
TIPC_LOW_IMPORTANCE 0
TLV320AIC23_CLKIN_SHIFT 6
TMEM_POOL_PAGESIZE_SHIFT 4
TMEM_VERSION_SHIFT 24
TMRDIV_SHIFT 0
TOMOYO_TYPE_EXECUTE 0
TP_ACPI_HOTKEYSCAN_FNEND 16
TP_ACPI_HOTKEYSCAN_FNHOME 15
TPD_BUFLEN_SHIFT 0
TPD_CC_SEGMENT_EN_SHIFT 3
TPD_CCSUM_EN_SHIFT 8
TPD_CCSUM_OFFSET_SHIFT 18
TPD_CCSUMOFFSET_SHIFT 24
TPD_CUST_CSUM_EN_SHIFT 3
TPD_EOP_SHIFT 0
TPD_EOP_SHIFT 31
TPD_ETH_TYPE_SHIFT 17
TPD_ETHTYPE_SHIFT 9
TPD_HADDR_SHIFT 5
TPD_HDRFLAG_SHIFT 18
TPD_INS_VL_TAG_SHIFT 2
TPD_INS_VTAG_SHIFT 15
TPD_IP_CSUM_SHIFT 5
TPD_IPHL_SHIFT 10
TPD_IPV4_PACKET_SHIFT 16
TPD_LSO_EN_SHIFT 12
TPD_LSO_VER_SHIFT 13
TPD_MSS_SHIFT 18
TPD_MSS_SHIFT 19
TPD_PLOADOFFSET_SHIFT 0
TPD_PLOADOFFSET_SHIFT 16
TPD_SEGMENT_EN_SHIFT 4
TPD_TCP_CSUM_SHIFT 6
TPD_TCPHDRLEN_SHIFT 14
TPD_TCPHDR_OFFSET_SHIFT 0
TPD_V4_IPHL_SHIFT 10
TPD_VLAN_SHIFT 16
TPD_VLANTAG_SHIFT 16
TPD_VL_TAGGED_SHIFT 8
```



```
TP_NVRAM_POS_LEVEL_BRIGHTNESS 0
TPS6105X_REG0_MODE_SHIFT 6
TPS6105X_REG0_VOLTAGE_SHIFT 4
TRANSPARENT_HUGEPAGE_FLAG 0
TRANSPARENT_HUGEPAGE_REQ_MADV_FLAG 1
TRB_MAX_BUFF_SHIFT 16
TRUEIDE_MWORD_DMA_TIMING_SHIFT 7
TRUEIDE_PIO_TIMING_SHIFT 4
TSIZ_NTD_SHIFT 8
TSIZ_PKTCNT_SHIFT 19
TSIZ_SCHINFO_SHIFT 0
TSIZ_SC_MC_PID_SHIFT 29
TSIZ_XFERSIZE_SHIFT 0
TSQ_THROTTLED 0
TST_SWITCH_PENDING 1
TST_SWITCH_WAIT 2
TT1_SHIFT 0
TT2_SHIFT 4
TTM_BO_PRIV_FLAG_MOVING 0
TT_OFFSET 2
TTY_CLOSING 7
TTY_DO_WRITE_WAKEUP 5
TTY_EXCLUSIVE 3
TTY_HUPPED 18
TTY_HUPPING 21
TTY_IO_ERROR 1
TTY_LDISC 9
TTY_LDISC_CHANGING 10
TTY_LDISC_HALTED 22
TTY_NO_WRITE_SPLIT 17
TTY_OTHER_CLOSED 2
TTYP_FLUSHPENDING 2
TTY_PTY_LOCK 16
TTY_PUSH 6
TTY_THROTTLED 0
TV_BURST_LEVEL_SHIFT 16
TV_DAC_CNTL_BGADJ__SHIFT 16
TV_DAC_CNTL_DACADJ__SHIFT 20
TV_HBLANK_END_SHIFT 16
TV_HBLANK_START_SHIFT 0
TV_HBURST_LEN_SHIFT 0
TV_HSYNC_END_SHIFT 16
TV_HTOTAL_SHIFT 0
TV_NBR_END_SHIFT 16
TVP5150_CROP_SHIFT 2
TV_SCDDA1_INC_SHIFT 0
TV_SCDDA2_INC_SHIFT 0
TV_SCDDA2_SIZE_SHIFT 16
TV_SCDDA3_INC_SHIFT 0
TV_SCDDA3_SIZE_SHIFT 16
TV_VBURST_END_F1_SHIFT 0
TV_VBURST_END_F2_SHIFT 0
TV_VBURST_END_F3_SHIFT 0
TV_VBURST_END_F4_SHIFT 0
TV_VBURST_START_F1_SHIFT 16
TV_VBURST_START_F2_SHIFT 16
TV_VBURST_START_F3_SHIFT 16
TV_VBURST_START_F4_SHIFT 16
TV_VEQ_LEN_SHIFT 16
TV_VEQ_START_F1_SHIFT 8
TV_VEQ_START_F2_SHIFT 0
TV_VI_END_F1_SHIFT 8
TV_VI_END_F2_SHIFT 0
TV_VSYNC_LEN_SHIFT 16
TV_VSYNC_START_F1_SHIFT 8
TV_VSYNC_START_F2_SHIFT 0
```

```
TW_IN_RESET 2
TWL4030_ROW_SHIFT 4
TWL6030_CFG_STATE_GRP_SHIFT 5
TWSIC0_SID_SHIFT 3
TWSIC1_ADDR_SHIFT 16
TW_USING_MSI 3
TXC_AMPDU_SHIFT 9
TXC_ARXCTL_RXPD0_LBN 12
TXC_ARXCTL_RXPD1_LBN 13
TXC_ARXCTL_RXPD2_LBN 14
TXC_ARXCTL_RXPD3_LBN 15
TXC_ATXCTL_TXPD0_LBN 12
TXC_ATXCTL_TXPD1_LBN 13
TXC_ATXCTL_TXPD2_LBN 14
TXC_ATXCTL_TXPD3_LBN 15
TXC_BIST_CTRL_ENAB_LBN 13
TXC_BIST_CTRL_STOP_LBN 14
TXC_BIST_CTRL_STRT_LBN 15
TXC_BIST_CTRL_TYPE_LBN 10
TXC_GLCMD_L01PD_LBN 5
TXC_GLCMD_L23PD_LBN 6
TXC_GLCMD_LMTSWRST_LBN 14
TXC_MCTL_RXLED_LBN 13
TXC_MCTL_TXLED_LBN 14
TX_CMD_SEC_SHIFT 6
TX_CMD_SEC_WEP_KEY_IDX_POS 6
TXC_MTDIABLO_CTRL_PMA_LOOP_LBN 10
TXD_LEN_SHIFT 16
TxDMABurstSizeShift 8
TXD_MSS_SHIFT 16
TxDRNT_shift 0
TXD_VLAN_TAG_SHIFT 0
TXFID_SEQ_SHIFT 5
TXFIFOCMD_FIFOSEL_SHIFT 8
TXFIFO_FIFOTOP_SHIFT 8
TxFILLT_shift 8
TX_FLOW_ON_BIT 1
TXHDR_IHL_SHIFT 52
TXHDR_L3START_SHIFT 48
TXHDR_L4START_SHIFT 40
TXHDR_L4STUFF_SHIFT 32
TXHDR_LEN_SHIFT 16
TXHDR_PAD_SHIFT 0
TxHiPriFIFOThreshShift 24
TX_HW_ATTR_OFST_RATE_POLICY 5
TX_HW_ATTR_OFST_SESSION_COUNTER 2
TX_JUMBO_TASK_TH_SHIFT 0
TX_LENGTHS_IPG_CRS_SHIFT 12
TX_LENGTHS_IPG_SHIFT 8
TX_LENGTHS_SLOT_TIME_SHIFT 0
TX_LOG_PAGE_VLD_FUNC_SHIFT 2
TxMXDMA_shift 20
TxPadLenShift 16
TXQ_CMD_SHIFT 29
TXQ_CTRL 2
TXQ_CTRL_TPD_BURST_NUM_SHIFT 0
TXQ_CTRL_TPD_FETCH_TH_SHIFT 8
TXQ_CTRL_TXF_BURST_NUM_SHIFT 16
TXQ_ETH 0
TXQ_OFLD 1
TXQ_PHY_SHIFT 12
TXQ_RUNNING 1
TXQ_SRS_SHIFT 20
TX_RNG_CFIG_LEN_SHIFT 48
TX_STATUS_BA_BMAP47_SHIFT 3
TXSTOPPED 1
```



```
TX_TPD_MIN_IPG_SHIFT 16
UAC_FORMAT_TYPE_I_ALAW 4
UAC_FORMAT_TYPE_I_IEEE_FLOAT 3
UAC_FORMAT_TYPE_I_MULAW 5
UAC_FORMAT_TYPE_I_PCM 1
UAC_FORMAT_TYPE_I_PCM8 2
UAC_FORMAT_TYPE_I_UNDEFINED 0
UBIFS_BLOCK_SHIFT 12
UBIFS_LPT_FANOUT_SHIFT 2
UDC_BITS_PER_BYTE_SHIFT 3
UDCCONR_AISN_S 19
UDCCONR_CN_S 25
UDCCONR_EN_S 15
UDCCONR_ET_S 13
UDCCONR_IN_S 22
UDCCONR_MPS_S 2
UDC_CSR_NE_ALT_SHIFT 15
UDC_CSR_NE_CFG_SHIFT 7
UDC_CSR_NE_DIR_SHIFT 4
UDC_CSR_NE_INTF_SHIFT 11
UDC_CSR_NE_MAX_PKT_SHIFT 19
UDC_CSR_NE_NUM_SHIFT 0
UDC_CSR_NE_TYPE_SHIFT 5
UDC_DEVCTL_BRLEN_SHIFT 16
UDC_DEVCTL_THLEN_SHIFT 24
UDC_EP0OUT_IX 16
UDC_EPCTL_ET_SHIFT 4
UDF_FLAG_BLOCKSIZE_SET 19
UDF_FLAG_GID_FORGET 13
UDF_FLAG_GID_IGNORE 14
UDF_FLAG_GID_SET 16
UDF_FLAG_LASTBLOCK_SET 18
UDF_FLAG_NLS_MAP 9
UDF_FLAG_SESSION_SET 17
UDF_FLAG_STRICT 5
UDF_FLAG_UID_FORGET 11
UDF_FLAG_UID_IGNORE 12
UDF_FLAG_UID_SET 15
UDF_FLAG_UNDELETE 6
UDF_FLAG_UNHIDE 7
UDF_FLAG_USE_AD_IN_ICB 3
UDF_FLAG_USE_SHORT_AD 2
UDF_FLAG_UTF8 10
UDI_SENDER_ID_SHIFT 8
UD_VECTOR 6
ULP_ACCEPT 9
ULP_F_CALL_PENDING 2
ULP_REJECT 10
ultra 2
ULTRA_DMA_TIMING_SHIFT 10
UNLOADING 15
Unmerged 2
UNPLUG_REMOTE 0
uPD98401_IM_I_SHIFT 24
uPD98401_PC_C_SHIFT 16
uPD98401_RXFP_ALERT_SHIFT 28
uPD98401_RXFP_BFSZ_SHIFT 24
uPD98401_RXFP_BTSZ_SHIFT 16
uPD98401_RXVC_POOL_SHIFT 16
uPD98401_TXVC_SHP_SHIFT 24
uPD98401_TXVC_VPI_SHIFT 16
UPROBE_COPY_INSN 0
USB_CAPTURE_RUNNING 0
USB_CDC_ETHERNET_TYPE 15
USB_CDC_HEADER_TYPE 0
USB_CDC_UNION_TYPE 6
```

```
USB_DETECT_ENABLE 3
USB_DEVICE_A_ALT_HNP_SUPPORT 5
USB_DEVICE_A_HNP_SUPPORT 4
USB_DEVICE_B_HNP_ENABLE 3
USB_DEVICE_REMOTE_WAKEUP 1
USB_DEVICE_SELF_POWERED 0
USB_DEV_STAT_U1_ENABLED 2
USB_DEV_STAT_U2_ENABLED 3
USB_EHCI_LOADED 2
USB_ENDPOINT_HALT 0
USB_FULL_SPEED 1
USB_HIGH_SPEED 2
USB_IN_ACK_RCVD 18
USB_IN_ACK_RCVD 3
USB_IN_NAK_SENT 19
USB_IN_NAK_SENT 4
USB_OHCI_LOADED 1
USB_OUT_ACK_SENT 1
USB_OUT_ACK_SENT 16
USB_OUT_NAK_SENT 2
USB_OUT_PING_NAK_SENT 17
USB_PLAYBACK_RUNNING 1
USB_PORT_FEAT_C_CONNECTION 16
USB_PORT_FEAT_C_RESET 20
USB_PORT_FEAT_C_SUSPEND 18
USB_PORT_FEAT_RESET 4
USB_PORT_FEAT_SUSPEND 2
USB_ROOT_PORT_WAKEUP_ENABLE 11
USB_ROOT_PORT_WAKEUP_ENABLE 5
USB_SOFT_RESET 1
USB_STALL_SENT 20
USB_STALL_SENT 5
USB_UHCI_LOADED 0
USE_DEGR_WFC_T 2
USER 4
USER_CONTROL_RES1_SHIFT 0
USER_ODIG_CTRL_GPIOS_SHIFT 5
USER_PMD_TX_CTL_TX_DAC_TXCK_SH 4
USER_PMD_TX_CTL_TX_DAC_TXD_SH 6
US_FLIDX_ABORTING 2
US_FLIDX_DISCONNECTING 3
US_FLIDX_READ10_WORKED 8
US_FLIDX_REDO_READ10 7
US_FLIDX_RESETTING 4
US_FLIDX_SCAN_PENDING 6
US_FLIDX_SG_ACTIVE 1
US_FLIDX_TIMED_OUT 5
US_FLIDX_URB_ACTIVE 0
USTORM_ISCSI_ST_CONTEXT_MAX_OUTSTANDING_R2TS_SHIFT 24
UV2_ACK_UNITS_SHFT 3
UV2_EXT_SHFT 33
UV_DESC_PSHIFT 49
UVH_IPI_INT_APIC_ID_SHFT 16
UVH_IPI_INT_SEND_SHFT 63
UVH_IPI_INT_VECTOR_SHFT 0
UVH_RH_GAM_GRU_OVERLAY_CONFIG_MMR_BASE_SHFT 28
UVH_RTC1_INT_CONFIG_APIC_ID_SHFT 32
UVH_RTC1_INT_CONFIG_VECTOR_SHFT 0
UV_PAYLOADQ_PNODE_SHIFT 49
UV_SW_ACK_NPENDING 8
V2_DQBLKSIZE_BITS 10
V4L2_COLORFX_BW 1
V4L2_COLORFX_NEGATIVE 3
V4L2_COLORFX_NONE 0
V4L2_COLORFX_SEPIA 2
V4L2_COLORFX_SKETCH 5
```



```
V4L2_FL_REGISTERED 0
V4L2_FL_USE_FH_PRIO 2
V4L2_FL_USES_V4L2_FH 1
V4L2_MPEG_AUDIO_AC3_BITRATE_256K 12
V4L2_MPEG_AUDIO_AC3_BITRATE_384K 14
V4L2_MPEG_AUDIO_DEC_PLAYBACK_AUTO 0
V4L2_MPEG_AUDIO_L2_BITRATE_256K 11
V4L2_MPEG_AUDIO_L2_BITRATE_384K 13
V4L2_MPEG_AUDIO_SAMPLING_FREQ_48000 1
V4L2_MPEG_STREAM_TYPE_MPEG2_TS 1
V4L2_MPEG_VIDEO_ENCODING_MPEG_2 1
VALID_BIT 31
VALUE_HASH_SHIFT 16
VBUS_INTERRUPT 2
VBUS_INTERRUPT 7
VBUS_INTERRUPT_ENABLE 2
VBUS_INTERRUPT_ENABLE 7
VBUS_PIN 0
VBUS_PIN 10
VCF_IDLE 2
VCF_RX 1
VCF_TX 0
VCPU_EXREG_CPL 20
VCPU_EXREG_CR3 18
VCPU_EXREG_PDPTR 17
VCPU_EXREG_RFLAGS 19
VCPU_EXREG_SEGMENTS 21
VCPU_REGS_RIP 16
VCPU_REGS_RSP 4
VDD1_VGAIN_SEL_SHIFT 6
VDD2_VGAIN_SEL_SHIFT 6
VERT_PANEL_SHIFT 12
VHOST_F_LOG_ALL 26
VHOST_NET_F_VIRTIO_NET_HDR 27
VHOST_VRING_F_LOG 0
VIA_MM_ALIGN_SHIFT 4
VIA_REG_AC97_CMD_SHIFT 16
VIA_REG_AC97_CODEC_ID_SHIFT 30
VIA_REG_AC97_DATA_SHIFT 0
VIA_STRFILT_CNT_SHIFT 16
VIDCFG_PIXFMT_SHIFT 18
V_INTR_PRIO_SHIFT 16
VIRTIO_CONSOLE_F_MULTIPORT 1
VIRTIO_NET_F_MRG_RXBUF 15
VIRTIO_RING_F_EVENT_IDX 29
VLAN_PRIO_SHIFT 13
VLAN_VALID 1
VMCB_DIRTY_MAX 11
VMWARE_PORT_CMD_LEGACY_X2APIC 3
VMWARE_PORT_CMD_VCPU_RESERVED 31
VMX_BASIC_MEM_TYPE_SHIFT 50
VMX_BASIC_VMCS_SIZE_SHIFT 32
VMX_EPT_GAW_EPTP_SHIFT 3
VMX_EPT_MT_EPTE_SHIFT 3
VMXNET3_RXD_BTYPE_SHIFT 14
VMXNET3_STATE_BIT_QUIESCED 1
VMXNET3_TXD_GEN_SHIFT 14
V_PARAM_SHIFT 1
VPD_CAP_VPD_ADDR_SHIFT 16
VP_DPC_NEEDED 14
VP_IDX_ACQUIRED 0
VP_SCR_NEEDED 4
VSC_MMIO_BAR 0
__VXGE_STATE_CARD_UP 1
W1_SLAVE_ACTIVE 0
WAKEUP_BEHAVIOR_RESTORE_CR4 1
```

```
4087 WAKEUP_BEHAVIOR_RESTORE_MISC_ENABLE 0
4088 WantReplacement 9
4089 WBD_ALPHA 6
4090 WDM_DISCONNECTING 2
4091 WDM_INT_STALL 5
4092 WDM_IN_USE 1
4093 WDM_OVERFLOW 10
4094 WDM_READ 4
4095 WDM_RESETTING 9
4096 WDM_RESPONDING 7
4097 WDM_SUSPENDING 8
4098 WDOG_ACTIVE 0
4099 WDOG_ALLOW_RELEASE 2
4100 WDOG_NO_WAY_OUT 3
4101 WDOG_UNREGISTERED 4
4102 WDT_FLAGS_ORPHAN 2
4103 WDT_OK_TO_CLOSE 1
4104 WDTS_EXPECTED 5
4105 WDTS_KEEPALIVE 2
4106 WDTS_USE_GP 4
4107 WFOI_MASK_SHIFT 4
4108 WH_OFS 20
4109 wil_status_dontscan 3
4110 wil_status_fwconnected 2
4111 wil_status_fwconnecting 1
4112 wil_status_fwready 0
4113 wil_status_irqen 5
4114 wil_status_reset_done 4
4115 WIMAX_ST_CONNECTED 8
4116 WIMAX_ST_CONNECTING 7
4117 WIMAX_ST_DOWN 1
4118 __WIMAX_ST_QUIESCING 2
4119 WIMAX_ST_RADIO_OFF 4
4120 WIMAX_ST_READY 5
4121 WIMAX_ST_SCANNING 6
4122 WIMAX_ST_UNINITIALIZED 3
4123 WL1271_FLAG_DUMMY_PACKET_PENDING 7
4124 WL1271_FLAG_FW_TX_BUSY 6
4125 WL1271_FLAG_IN_ELP 3
4126 WL1271_FLAG_INTENDED_FW_RECOVERY 13
4127 WL1271_FLAG_IRQ_RUNNING 5
4128 WL1271_FLAG_PENDING_WORK 9
4129 WL1271_FLAG_RECOVERY_IN_PROGRESS 11
4130 WL1271_FLAG_SOFT_GEMINI 10
4131 WL1271_FLAG_SUSPENDED 8
4132 WL1271_FLAG_TX_PENDING 2
4133 WL1271_FLAG_VIF_CHANGE_IN_PROGRESS 12
4134 WLAN_ENABLED 7
4135 WLAN_STA_ASSOC 1
4136 WLAN_STA_AUTH 0
4137 WLAN_STA_AUTHORIZED 3
4138 WLVIF_FLAG_AP_PROBE_RESP_SET 10
4139 WLVIF_FLAG_AP_STARTED 4
4140 WLVIF_FLAG_CS_PROGRESS 9
4141 WLVIF_FLAG_IBSS_JOINED 3
4142 WLVIF_FLAG_INITIALIZED 0
4143 WLVIF_FLAG_IN_PS 5
4144 WLVIF_FLAG_IN_USE 11
4145 WLVIF_FLAG_RX_STREAMING_STARTED 7
4146 WLVIF_FLAG_STA_ASSOCIATED 1
4147 WLVIF_FLAG_STA_AUTHORIZED 2
4148 WM0_PIPE_PLANE_SHIFT 16
4149 WM0_PIPE_SPRITE_SHIFT 8
4150 WM2200_AIF1TX_WL_SHIFT 8
4151 WM2200_FLL_CLK_REF_DIV_SHIFT 4
4152 WM2200_FLL_OUTDIV_SHIFT 8
```



```
4153 WM2200_IN1_DMIC_SUP_SHIFT 11
4154 WM2200_IN1_MODE_SHIFT 9
4155 WM2200_MICB1_LVL_SHIFT 2
4156 WM2200_SYSCLK_FREQ_SHIFT 8
4157 WM5100_ACCDET_BIAS_SRC_SHIFT 14
4158 WM5100_ACCDET_BIAS_STARTTIME_SHIFT 12
4159 WM5100_ACCDET_RATE_SHIFT 8
4160 WM5100_ACCDET_SRC_SHIFT 13
4161 WM5100_AIF1TX_WL_SHIFT 8
4162 WM5100_FLL1_OUTDIV_SHIFT 8
4163 WM5100_FLL1_REFCLK_DIV_SHIFT 6
4164 WM5100_GP1_DIR_SHIFT 15
4165 WM5100_GP1_FN_SHIFT 0
4166 WM5100_GP1_LVL_SHIFT 6
4167 WM5100_HPCOM_SRC_SHIFT 9
4168 WM5100_IN1_DMIC_SUP_SHIFT 11
4169 WM5100_IN1_MODE_SHIFT 9
4170 WM5100_SYSCLK_FREQ_SHIFT 8
4171 WM831X_CLKOUT_SRC_SHIFT 0
4172 WM831X_DC1_DVS_SRC_SHIFT 11
4173 WM831X_DC1_HC_THR_SHIFT 4
4174 WM831X_DC1_ON_MODE_SHIFT 8
4175 WM831X_LED_DUR_SHIFT 2
4176 WM831X_LED_MODE_SHIFT 8
4177 WM831X_LED_SRC_SHIFT 14
4178 WM831X_WDOG_PRIMACT_SHIFT 4
4179 WM831X_WDOG_RST_SRC_SHIFT 13
4180 WM831X_WDOG_SECACT_SHIFT 8
4181 WM8350_DC1_ENSLOT_SHIFT 10
4182 WM8350_DC1_ERRACT_SHIFT 14
4183 WM8350_DC1_SDSLOT_SHIFT 6
4184 WM8350_DC2_FBSRC_SHIFT 0
4185 WM8350_DC2_HIB_MODE_SHIFT 12
4186 WM8350_DC2_ILIM_SHIFT 6
4187 WM8350_DC2_MODE_SHIFT 14
4188 WM8350_DC2_RMP_SHIFT 3
4189 WM8350_DC5_FBSRC_SHIFT 0
4190 WM8350_DC5_HIB_MODE_SHIFT 12
4191 WM8350_DC5_ILIM_SHIFT 6
4192 WM8350_DC5_MODE_SHIFT 14
4193 WM8350_DC5_RMP_SHIFT 3
4194 WM8350_OUT1L_VOL_SHIFT 2
4195 WM8350_OUT1R_VOL_SHIFT 2
4196 WM8350_RTC_ALMDAY_SHIFT 8
4197 WM8350_RTC_ALMMINS_SHIFT 8
4198 WM8350_RTC_ALMMTH_SHIFT 8
4199 WM8350_RTC_DAY_SHIFT 8
4200 WM8350_RTC_MINS_SHIFT 8
4201 WM8350_RTC_MTH_SHIFT 8
4202 WM8350_RTC_YHUNDREDS_SHIFT 8
4203 WM8400_AINLMUX_PWR 1
4204 WM8400_AINRMUX_PWR 3
4205 WM8400_FLL_REF_FREQ_SHIFT 12
4206 WM8400_INMIXL_PWR 0
4207 WM8400_INMIXR_PWR 2
4208 WM8737_SR_SHIFT 1
4209 WM8903_CLK_SYS_MODE_SHIFT 8
4210 WM8903_CLK_SYS_RATE_SHIFT 10
4211 WM8903_DEEMPH_SHIFT 1
4212 WM8903_GP1_FN_SHIFT 8
4213 WM8903_GP1_LVL_SHIFT 4
4214 WM8903_GP2_LVL_SHIFT 4
4215 WM8903_ISEL_SHIFT 2
4216 WM8903_VMID_SOFT_SHIFT 3
4217 WM8904_CLK_SYS_RATE_SHIFT 10
4218 WM8904_DCS_TRIG_STARTUP_0_SHIFT 4
```

```
WM8904_DEEMPH_SHIFT 1
WM8904_FLL_CLK_REF_DIV_SHIFT 3
WM8904_FLL_FRATIO_SHIFT 0
WM8904_FLL_N_SHIFT 5
WM8904_FLL_OUTDIV_SHIFT 8
WM8904_ISEL_SHIFT 2
WM8904_SAMPLE_RATE_SHIFT 0
WM8904_VMID_RES_SHIFT 1
WM8955_DEEMPH_SHIFT 1
WM8955_N_SHIFT 5
WM8955_SR_SHIFT 1
WM8955_VMIDSEL_SHIFT 7
WM8955_VREF_SHIFT 6
WM8955_VSEL_SHIFT 6
WM8958_DSP2CLK_SRC_SHIFT 12
WM8958_MBC_SEL_SHIFT 4
WM8958_MICD_BIAS_STARTTIME_SHIFT 12
WM8958_MICD_RATE_SHIFT 8
WM8961_CLK_SYS_RATE_SHIFT 1
WM8961_VMIDSEL_SHIFT 7
WM8961_WL_SHIFT 2
WM8962_BEEP_RATE_SHIFT 1
WM8962_CLKOUT2_SEL_SHIFT 5
WM8962_CLKOUT3_SEL_SHIFT 3
WM8962_FLL_OUTDIV_SHIFT 3
WM8962_FLL_REFCLK_SRC_SHIFT 5
WM8962_GP2_FN_SHIFT 0
WM8962_GP2_LVL_SHIFT 6
WM8962_SYSCLK_RATE_SHIFT 1
WM8962_SYSCLK_SRC_SHIFT 9
WM8983_BCLKDIV_SHIFT 2
WM8983_BCP_SHIFT 8
WM8983_EQ3DMODE_SHIFT 8
WM8983_FMT_SHIFT 3
WM8983_LRCP_SHIFT 7
WM8983_MCLKDIV_SHIFT 5
WM8983_MS_SHIFT 0
WM8983_PLL_PRESCALE_SHIFT 4
WM8983_SOFTMUTE_SHIFT 6
WM8983_SR_SHIFT 1
WM8983_VMIDSEL_SHIFT 0
WM8983_WL_SHIFT 5
WM8985_BCLKDIV_SHIFT 2
WM8985_BCP_SHIFT 8
WM8985_EQ3DMODE_SHIFT 8
WM8985_FMT_SHIFT 3
WM8985_LRP_SHIFT 7
WM8985_MCLKDIV_SHIFT 5
WM8985_MS_SHIFT 0
WM8985_PLL_PRESCALE_SHIFT 4
WM8985_SOFTMUTE_SHIFT 6
WM8985_SR_SHIFT 1
WM8985_VMIDSEL_SHIFT 0
WM8985_WL_SHIFT 5
WM8990_AINLMUX_PWR_BIT 1
WM8990_AINRMUX_PWR_BIT 3
WM8990_INMIXL_PWR_BIT 0
WM8990_INMIXR_PWR_BIT 2
WM8991_AINLMUX_PWR_BIT 1
WM8991_AINRMUX_PWR_BIT 3
WM8991_INMIXL_PWR_BIT 0
WM8991_INMIXR_PWR_BIT 2
WM8993_BCLK_DIV_SHIFT 1
WM8993_CLK_SYS_RATE_SHIFT 1
WM8993_DCS_DAC_WR_VAL_1_SHIFT 8
WM8993_DCS_SERIES_NO_01_SHIFT 5
```



```
WM8993_FLL_CLK_REF_DIV_SHIFT 3
WM8993_FLL_FRATIO_SHIFT 0
WM8993_FLL_N_SHIFT 5
WM8993_FLL_OUTDIV_SHIFT 8
WM8993_JD_SCTHR_SHIFT 6
WM8993_JD_THR_SHIFT 4
WM8993_MICB2_LVL_SHIFT 1
WM8993_SAMPLE_RATE_SHIFT 7
WM8994_AIF1_BCLK_DIV_SHIFT 4
WM8994_AIF1DAC1_3D_GAIN_SHIFT 9
WM8994_AIF1DAC2_3D_GAIN_SHIFT 9
WM8994_AIF1_SR_SHIFT 4
WM8994_AIF2DAC_3D_GAIN_SHIFT 9
WM8994_CP_DYN_SRC_SEL_SHIFT 8
WM8994_FLL1_FRATIO_SHIFT 0
WM8994_FLL1_FRC_NCO_SHIFT 6
WM8994_FLL1_N_SHIFT 5
WM8994_FLL1_OUTDIV_SHIFT 8
WM8994_FLL1_REFCLK_DIV_SHIFT 3
WM8994_VMID_RAMP_SHIFT 5
WM8995_AIF1_BCLK_DIV_SHIFT 4
WM8995_AIF1DAC1_MUTE_SHIFT 9
WM8995_AIF1_FMT_SHIFT 3
WM8995_AIF1_SR_SHIFT 4
WM8995_AIF1_WL_SHIFT 5
WM8995_CP_DYN_SRC_SEL_SHIFT 8
WM8995_FLL1_FRATIO_SHIFT 0
WM8995_FLL1_N_SHIFT 5
WM8995_FLL1_OUTDIV_SHIFT 8
WM8995_FLL1_REFCLK_DIV_SHIFT 3
WM8996_AIF1RX_CHAN0_SLOTS_SHIFT 6
WM8996_AIF1RX_CHAN1_SLOTS_SHIFT 6
WM8996_AIF1RX_CHAN2_SLOTS_SHIFT 6
WM8996_AIF1RX_CHAN3_SLOTS_SHIFT 6
WM8996_AIF1RX_CHAN4_SLOTS_SHIFT 6
WM8996_AIF1RX_CHAN5_SLOTS_SHIFT 6
WM8996_AIF1TX_CHAN0_SLOTS_SHIFT 6
WM8996_AIF1TX_CHAN1_SLOTS_SHIFT 6
WM8996_AIF1TX_CHAN2_SLOTS_SHIFT 6
WM8996_AIF1TX_CHAN3_SLOTS_SHIFT 6
WM8996_AIF1TX_CHAN4_SLOTS_SHIFT 6
WM8996_AIF1TX_CHAN5_SLOTS_SHIFT 6
WM8996_AIF1TX_WL_SHIFT 8
WM8996_AIF2RX_CHAN0_SLOTS_SHIFT 6
WM8996_AIF2RX_CHAN1_SLOTS_SHIFT 6
WM8996_AIF2TX_CHAN0_SLOTS_SHIFT 6
WM8996_DCS_TRIG_STARTUP_0_SHIFT 4
WM8996_FLL_LFSR_SEL_SHIFT 1
WM8996_FLL_N_SHIFT 5
WM8996_FLL_OUTDIV_SHIFT 8
WM8996_FLL_REFCLK_DIV_SHIFT 3
WM8996_FLL_REF_FREQ_SHIFT 2
WM8996_GP1_DIR_SHIFT 15
WM8996_GP1_FN_SHIFT 0
WM8996_GP1_LVL_SHIFT 6
WM8996_INL_MODE_SHIFT 2
WM8996_MICD_BIAS_STARTTIME_SHIFT 12
WM8996_MICD_RATE_SHIFT 8
WM8996_SYSCLK_SRC_SHIFT 3
WM9081_AIFDAC_TDM_MODE_SHIFT 2
WM9081_CLK_SYS_RATE_SHIFT 4
WM9081_FLL_CLK_REF_DIV_SHIFT 3
WM9081_FLL_FRATIO_SHIFT 0
WM9081_FLL_N_SHIFT 5
WM9081_FLL_OUTDIV_SHIFT 8
WM9081_SAMPLE_RATE_SHIFT 0
```

```
WM9090_VMID_RES_SHIFT 1
WMI_CTRL_EP_FULL 2
WMI_DATA_HDR_DATA_TYPE_SHIFT 6
WMI_DATA_HDR_MSG_TYPE_SHIFT 0
WMI_ENABLED 0
WMI_RC_RX_STBC_FLAG_S 6
WMI_READY 1
WMM_AC_BK 1
WMM_ENABLED 2
WMM_NUM_AC 4
WM_SW_DAC 0
WORK_DONE_BIT 1
WORK_ENABLE 3
WORK_HIGH_PRIO_BIT 3
WORK_LINK_DOWN 2
WORK_LINK_UP 1
WORK_OFFQ_POOL_SHIFT 6
WORK_SET_MULTICAST_LIST 4
WORK_STRUCT_COLOR_BITS 4
WORK_STRUCT_COLOR_SHIFT 5
Wpending 8
WriteErrorSeen 6
WriteMostly 3
WRITE_REMOTE_AMP_ASSOC 3
WS_OFS 28
WSPI_CMD_BYTE_LENGTH_OFFSET 17
WST_OFS 22
WTSR_EN_SHIFT 6
WTSRT_SHIFT 0
X25_ACCPT_APPRV_FLAG 2
X25_Q_BIT_FLAG 0
XEN_MCE_OVERFLOW 0
_XEN_PCIB_active 2
_XEN_PCIB_AERHANDLER 1
_XEN_PCIF_active 0
XFS_BLF_SHIFT 7
XFS_EFI_RECOVERED 1
XFS_ILOCK_SHIFT 24
XFS_IOLOCK_SHIFT 16
XFTS_CHANNEL_SHIFT 8
XFTS_FBRRTS_FT_SHIFT 4
XFTS_RTS_FT_SHIFT 2
XGMAC_FLOW_CTRL_PT_SHIFT 16
XMAC_IPG_IPG_MII_GMII_SHIFT 8
XMAC_IPG_IPG_XGMII_SHIFT 0
XMAC_MIN_RX_MIN_PKT_SIZE_SHFT 20
XMAC_MIN_TX_MIN_PKT_SIZE_SHFT 0
XMIT_BUF_ONE_READY 6
XMIT_BUF_TWO_READY 7
XMIT_SENDING_READY 8
XMIT_WAITING 8
XMIT_WAKEUP 0
XMIT_WAKEUP 2
XPRT_CLOSE_WAIT 3
XPRT_CLOSING 6
XPRT_CONGESTED 9
XPRT_CONNECTION_ABORT 7
XPRT_CONNECTION_CLOSE 8
XPT_BUSY 0
XPT_CACHE_AUTH 12
XPT_CHNGBUF 7
XPT_CLOSE 2
XPT_CONN 1
XPT_DATA 3
XPT_DEFERRED 8
XPT_DETACHED 10
```



```
XPT_LISTENER 11
XPT_TEMP 4
XSTORM_COMMON_CONTEXT_SECTION_PBF_PORT_SHIFT 1
XSTORM_COMMON_CONTEXT_SECTION_PHYSQ_INITIALIZED_SHIFT 0
XT_DSCP_SHIFT 2
XY_BUF_OFFSET 4
ZCACHE_DSTMEM_ORDER 1
ZD_DEVICE_RUNNING 0
ZR36057_FHAP_NAX 16
ZR36057_FHAP_PAX 0
ZR36057_FVAP_NAY 16
ZR36057_FVAP_PAY 0
ZR36057_HSP_HsyncStart 16
ZR36057_HSP_LineTot 0
ZR36057_JCGI_JPEGuestID 4
ZR36057_JCGI_JPEGuestReg 0
ZR36057_VDCR_MinPix 24
ZR36057_VDCR_VidWinHt 12
ZR36057_VDCR_VidWinWid 0
ZR36057_VFEHCR_HEnd 0
ZR36057_VFEHCR_HStart 10
ZR36057_VFESPFR_DispMode 6
ZR36057_VFESPFR_HFilter 21
ZR36057_VFESPFR_HorDcm 14
ZR36057_VFESPFR_VerDcm 8
ZR36057_VFEVCR_VEnd 0
ZR36057_VFEVCR_VStart 10
ZR36057_VSP_FrmTot 0
ZR36057_VSP_VsyncSize 16
ZR36057_VSSFGR_DispStride 16
```

```
**********************************************************
     230 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.bit_shifters.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// tvaudio.c deliberately reuses these in a non conventional way
V4L2_TUNER_MODE_MONO 0
V4L2_TUNER_MODE_STEREO 1
V4L2_TUNER_MODE_LANG2 2
V4L2_TUNER_MODE_SAP 2
V4L2_TUNER_MODE_LANG1 3
V4L2_TUNER_MODE_LANG1_LANG2 4

MBX_INTERRUPT 1
```

**new/usr/src/tools/smatch/src/smatch_data/kernel.check_string_condition.ignore 1**

```
*********************************************************
      12 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.check_string_condition.ignore
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
*********************************************************
   1 TRACE_EVENT
```

```
**********************************************************
     179 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.clears_argument
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 copy_from_user 0
   2 __copy_from_user 0
   3 copy_user_generic 0
   4 loop_get_status 1
   5 loop_info64_to_old 1
   6 ib_copy_qp_attr_to_user 0
   7 tcp_get_info 1
   8 snd_rawmidi_info_select 1
   9 t3e3_if_config 3
```

```
**********************************************************
     972 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.common_functions
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
printk
memcpy
dev_err
kfree
writel
memset
lockdep_rcu_suspicious
variable_test_bit
mutex_unlock
spin_unlock_irqrestore
nv_wr32
dev_printk
no_printk
spin_unlock
warn_slowpath_null
_raw_spin_unlock_irqrestore
spinlock_check
mutex_lock_nested
outb
r100_mm_wreg
snprintf
sprintf
readl
usb_control_msg
kmalloc
warn_slowpath_fmt
dev_set_drvdata
constant_test_bit
current_thread_info
__udelay
i2c_transfer
__snd_printk
__fswab32
_raw_spin_unlock
seq_printf
get_current
__kmalloc
cit_write_reg
spin_lock
__list_add
trace_define_field
set_bit
_raw_spin_lock_irqsave
debugfs_create_file
list_empty
list_add_tail
dev_warn
netdev_priv
_raw_spin_lock
dev_get_drvdata
atomic_read
nv_mthd
INIT_LIST_HEAD
mod_phy_reg
_dev_info
clear_bit
__fswab16
spin_unlock_irq
iowrite32
__raw_spin_lock_init
```



```
debug_lockdep_rcu_enabled
stv090x_write_reg
kzalloc
i2c_master_send
nv_rd32
IS_ERR
b43_phy_write
lock_is_held
task_pid_nr
spin_unlock_bh
__wake_up
drm_ut_debug_printk
reg_w
writeb
__create_pipe
```

```
**********************************************************
     765 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.dev_queue_xmit
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
br_dev_queue_push_xmit 0
neigh_compat_output 0
fcoe_fip_send 1
eql_slave_xmit 0
irlap_queue_xmit 1
can_send 0
snap_request 1
llc_sap_action_send_xid_c 1
llc_sap_action_send_ui 1
x25_transmit_link 0
pEII_request 1
br_nf_dev_queue_xmit 0
pppoe_xmit 1
lapbeth_data_transmit 1
irlap_send_i_frame 1
aarp_send_ddp 1
x25_send_iframe 1
ipddp_xmit 0
pipe_skb_send 1
ax25_queue_xmit 0
llc_station_send_pdu 0
macvlan_start_xmit 0
dsa_xmit 0
pn_skb_send 1
pspoll_send_buffered 2
dev_queue_xmit 0
irlap_send_ui_frame 1
p8022_request 1
hostap_mgmt_start_xmit 0
edsa_xmit 0
nes_nic_cm_xmit 0
__pppoe_xmit 1
pvc_xmit 0
p8023_request 1
macvlan_queue_xmit 0
llc_build_and_send_ui_pkt 1
x25_send_frame 0
bpq_xmit 0
pppoe_rcv_core 1
pn_send 0
llc_sap_action_send_test_c 1
phonet_rcv 0
```

```
**********************************************************
     809 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.dma_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of DMA function and buffer parameters.
// generated by 'gen_dma_funcs.sh'
alauda_get_redu_data 2
alauda_read_block 4
alauda_read_block_raw 4
alauda_write_block 2
brcmf_usb_dl_send_bulk 1
datafab_bulk_read 1
datafab_bulk_write 1
ene_get_card_type 2
ene_send_scsi_cmd 2
fill_isoc_urb 3
ft1000_control 6
jumpshot_bulk_read 1
jumpshot_bulk_write 1
ms_lib_read_extrablock 4
ms_read_copyblock 5
ms_read_readpage 3
mts_int_submit_urb 2
rts51x_bulk_transfer_buf 2
rts51x_bulk_transport 4
rts51x_bulk_transport_special 4
rts51x_transfer_data 2
rts51x_transfer_data_rcc 2
sddr55_bulk_transport 2
sisusb_bulkin_msg 2
sisusb_bulkout_msg 3
usbat_bulk_read 1
usbat_bulk_write 1
__usb_control_msg 6
usb_control_msg 6
usb_fill_bulk_urb 3
usb_stor_bulk_transfer_buf 2
usb_stor_bulk_transfer_sg 2
vub300_usb_bulk_msg 2
```

```
**********************************************************
    1308 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.expects_err_ptr
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of functions which expect an ERR_PTR.
// generated by 'gen_expects_err_ptr.sh'
bnx2fc_flogi_resp 1
bnx2fc_logo_resp 1
cfg80211_unlock_rdev 0
cifs_get_tlink 0
cifs_put_tlink 0
class_destroy 0
clusterip_seq_stop 1
device_create_vargs 0
d_obtain_alias 0
do_readlink 2
d_splice_alias 0
fc_disc_gpn_ft_resp 1
fc_disc_gpn_id_resp 1
fc_els_resp_type 0
fc_exch_rrq_resp 1
fc_fcp_rec_resp 1
fc_fcp_recv 1
fc_fcp_srr_resp 1
fc_lport_bsg_resp 1
fc_lport_flogi_resp 1
fc_lport_logo_resp 1
fc_lport_ms_resp 1
fc_lport_ns_resp 1
fc_lport_scr_resp 1
fcoe_flogi_resp 1
fcoe_logo_resp 1
fc_rport_adisc_resp 1
fc_rport_error 1
fc_rport_flogi_resp 1
fc_rport_logo_resp 1
fc_rport_plogi_resp 1
fc_rport_prli_resp 1
fc_rport_rtv_resp 1
fc_tm_done 1
free_link 0
ft_recv_seq 1
ieee80211_wep_decrypt_data 0
ieee80211_wep_encrypt_data 0
is_bad 0
is_error_page 0
kmemleak_alloc 0
kmemleak_alloc_percpu 0
kmemleak_free 0
kmemleak_free_part 0
kmemleak_free_percpu 0
kmemleak_ignore 0
kmemleak_no_scan 0
kmemleak_not_leak 0
kmemleak_scan_area 0
kmemleak_seq_stop 1
nfs3_cache_acls 1
nfs3_cache_acls 2
nfs_follow_remote_path 0
ntfs_runlists_merge 0
ntfs_runlists_merge 1
osduld_put_device 0
PTR_RET 0
quota_quotaon 4
```



```
regulator_put 0
securityfs_remove 0
tty_read 0
__vfs_follow_link 1
vfs_readlink 3
wait_on_page_read 0
xs_error 0
```

```
**********************************************************
   10619 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.frees_argument
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of functions and the argument they free.
// generated by 'gen_frees_list.sh'
__aarp_expire 0
acpi_bus_hot_remove_device 0
acpi_ds_delete_walk_state 0
acpi_ev_delete_gpe_xrupt 0
acpi_os_delete_lock 0
acpi_ut_delete_internal_object_list 0
ad714x_remove 0
adu_delete 0
adxl34x_remove 0
ahc_alloc 1
ahc_dma_tag_destroy 1
ahc_free 0
ahd_dma_tag_destroy 1
ahd_free 0
amixer_mgr_destroy 0
ap_free_sta 1
argv_free 0
ast_connector_destroy 0
ast_crtc_destroy 0
ast_encoder_destroy 0
ast_i2c_destroy 0
ast_ttm_backend_destroy 0
asus_cleanup_pci_hotplug 0
ata_sas_port_destroy 0
ath10k_htc_destroy 0
ath10k_htt_detach 0
ath6kl_wmi_shutdown 0
ath9k_htc_hw_free 0
ath_gen_timer_free 1
ath_mci_del_profile 2
atm_mpoa_delete_qos 0
atom_destroy 0
au0828_usb_release 0
audit_free_rule 0
autofs4_free_ino 0
__ax25_put_route 0
b1_free_card 0
b43_destroy_pioqueue_rx 0
balloon_devinfo_free 0
balloon_mapping_free 0
bar_release 2
batadv_hash_destroy 0
bcm_remove_op 0
bio_free_map_data 0
bl_free_lseg 0
blkg_free 0
blk_trace_free 0
block2mtd_free_device 0
bnx2fc_hba_destroy 0
bnx2x_exe_queue_free_elem 1
brcmf_free_vif 1
brcms_c_ampdu_detach 0
brcms_c_antsel_detach 0
brcms_c_bsscfg_mfree 0
brcms_c_channel_mgr_detach 0
brcms_ucode_free_buf 0
bsg_destroy_job 0
btrfs_free_block_rsv 1
```



```
btrfsic_block_free 0
btrfsic_block_link_free 0
btrfsic_dev_state_free 0
btrfsic_stack_frame_free 0
c2_dealloc_ucontext 0
cache_destroy_net 0
call_usermodehelper_freeinfo 0
cat_destroy 0
cat_destroy 1
cciss_free_sg_chain_blocks 0
cdc_ncm_free 0
ceph_auth_destroy 0
ceph_mdsmap_destroy 0
ceph_put_page_vector 0
ceph_release_page_vector 0
cffrml_free 0
cfg80211_dev_free 0
channel_detector_exit 1
cipso_v4_doi_free 0
cirrus_connector_destroy 0
cirrus_ttm_backend_destroy 0
class_compat_unregister 0
class_create_release 0
class_osdblk_release 0
class_pktcdvd_release 0
clkdev_drop 0
close_bwqcam 0
close_cqcam 0
cls_destroy 0
cls_destroy 1
cm109_usb_cleanup 0
cma3000_exit 0
cm_create_port_fs 0
cm_free_work 0
common_destroy 0
common_destroy 1
complete_agent_reset_write_no_wait 4
cond_destroy_bool 0
cond_destroy_bool 1
context_free 0
crush_destroy_bucket_list 0
crush_destroy_bucket_straw 0
crush_destroy_bucket_tree 0
crush_destroy_bucket_uniform 0
crystalhd_delete_dioq 1
CsrWifiNmeApPfree 0
CsrWifiRouterCtrlPfree 0
CsrWifiRouterPfree 0
CsrWifiRouterTransportSerialiseAndSend 1
CsrWifiSmePfree 0
ct_atc_destroy 0
ct_mixer_destroy 0
ct_timer_free 0
cuse_gendev_release 0
cxio_hal_destroy_resource 0
cyttsp_remove 0
dapm_free_path 0
db9_remove 0
dca_free_domain 0
delete_adapter 0
delete_attr_set 0
delete_port 0
destroy_8023_client 0
destroy_av 1
destroy_cache_args 0
destroy_EII_client 0
```

```
destroy_header_iter 0
destroy_htc_txctrl_packet 0
destroy_log_context 0
destroy_mount_options 0
destroy_trace_option_files 0
devinfo_seq_stop 1
diolan_u2c_free 0
dlm_free_lvb 0
dlm_free_pagevec 0
dl_seq_next 1
dma_buf_detach 1
dma_unpin_iovec_pages 0
dm_bio_prison_destroy 0
dm_block_manager_destroy 0
dm_deferred_set_destroy 0
dm_free_md_mempools 0
dmi_entry_free 0
dm_io_client_destroy 0
doc_release_device 0
do_sync_work 0
do_thaw_all 0
drbd_free_bc 0
drr_destroy_class 1
dummy_free 0
dvb_unregister_device 0
dynamic_kobj_release 0
_edac_mc_free 0
eeepc_cleanup_pci_hotplug 0
efivarfs_destroy 0
erase_callback 0
eventfd_free_ctx 0
ext4_remove_li_request 0
extcon_dev_release 0
fb_destroy_modedb 0
fcoe_sysfs_fcf_del 0
fib_free_table 0
filenametr_destroy 0
filenametr_destroy 1
flexcop_device_kfree 0
flow_destroy_filter 1
flush_entry_free 0
__fmc_sdb_free 0
force_clean_group 0
__fprog_destroy 0
framebuffer_release 0
free_buffer_page 0
free_buf_info 1
free_cmd 0
free_cmd_box 1
free_conn 0
free_context 0
free_cpumask_var 0
free_dca_provider 0
free_device_state 0
free_dev_ioctl 0
free_event_entry 0
free_fake_cpuc 0
free_flex_gd 0
free_fw_cache_entry 0
free_hba 0
free_i7core_dev 0
free_imon_context 0
free_ipath 0
free_irq_cfg 1
free_link_state 0
free_local_pdev 0
```



```
free_log_tree 1
free_map_info 0
free_msg 0
free_ncci 1
free_param_target 0
free_partition 0
free_partitions 0
free_pasid_state 0
free_pgpath 0
free_pipe_info 0
free_pl 0
free_plci 1
free_sbridge_dev 0
free_sched_domains 0
free_sched_group 0
free_symbol_cache 0
free_sysfs_super_info 0
free_trace_uprobe 0
free_trial_cpuset 0
free_tty_struct 0
free_usb_address 1
free_watch_adapter 0
__free_xattr 0
fsnotify_final_destroy_group 0
fs_path_free 0
ftl_erase_callback 0
ftrace_free_entry 0
fuse_file_free 0
fuse_free_conn 0
fwnet_pd_delete 0
fwnet_receive_packet 1
garp_attr_destroy 1
gcov_iter_free 0
gc_remove 0
gen_pool_destroy 0
gf128mul_free_4k 0
gf128mul_free_64k 0
gntdev_free_map 0
gred_destroy_vq 0
groups_free 0
gsm_free_mux 0
gss_do_free_ctx 0
gss_free_cred 0
hashtab_destroy 0
hfs_bnode_free 0
hfsplus_bnode_free 0
hid_free_report 0
hidinput_cleanup_hidinput 1
hsi_free_msg 0
hso_free_shared_int 0
htb_destroy_class 1
i2c_tiny_usb_free 0
i2o_block_device_free 0
i2o_exec_wait_free 0
i915_error_object_free 0
ib_dealloc_device 0
ibft_kobj_release 0
ibmpex_bmc_delete 0
idmouse_delete 0
ieee80211_key_free_common 0
if_spi_h2c 1
iio_buffer_remove_and_free_scan_dev_attr 1
iio_channel_release_all_cb 0
iio_dealloc_pollfunc 0
iio_device_remove_and_free_read_attr 1
intel_crt_destroy 0
```

```
intel_dvo_destroy 0
intel_hdmi_destroy 0
intel_set_config_free 0
intel_tv_destroy 0
ioeventfd_release 0
iommu_domain_free 0
iowarrior_delete 0
ip6mr_free_table 0
ipack_bus_unregister 0
ipath_user_sdma_queue_destroy 0
ipmr_free_table 0
__ipoctal_remove 0
ip_options_get_finish 2
ip_vs_app_inc_destroy 0
ip_vs_dest_dst_free 0
ip_vs_service_free 0
ip_vs_sync_buff_release 0
ipwireless_network_free 0
__ircomm_close 0
__ircomm_tty_cleanup 0
__irda_task_delete 0
__iriap_close 0
__irias_delete_attrib 0
__irias_delete_object 0
irias_delete_value 0
__irlap_close 0
__irlmp_close_lsap 0
isdn_v110_close 0
iwch_free_fastreg_pbl 0
iwl_trans_pcie_free 0
jffs2_free_full_dirent 0
kbd_disconnect 0
kvm_arch_free_vm 0
lapb_free_cb 0
libipw_txb_free 0
loop_remove 0
lpfc_els_free_bpl 1
lpfc_els_hbq_free 1
lpfc_free_ct_rsp 1
lpfc_hba_free 0
lpfc_sli4_rb_free 1
mac_destroy 0
mac_hid_emumouse_disconnect 0
mce_device_release 0
mdiobus_free 0
md_unplug 0
mei_io_cb_free 0
mempool_kfree 0
__mesh_table_free 0
mga_connector_destroy 0
mgag200_i2c_destroy 0
mgag200_ttm_backend_destroy 0
minstrel_free 0
mlx4_en_filter_free 0
mmc_free_ext_csd 0
mmc_test_free_mem 0
move_node 2
mpi_free_limb_space 0
mpt_adapter_dispose 0
mptsas_port_delete 1
mrp_attr_destroy 1
mthca_free_mailbox 1
mv88e1xxx_destroy 0
mv88x201x_destroy 0
mwifiex_unregister 0
my3126_destroy 0
```



```
netlink_consume_callback 0
netlink_destroy_callback 0
nfc_llc_free 0
nfc_llcp_free_sdp_tlv 0
nfc_mei_phy_free 0
nfs3_free_createdata 0
nfs41_free_stateid_release 0
nfs4_delegreturn_release 0
nfs4_free_open_state 0
nfs4_free_pages 0
nfs4_release_lockowner_release 0
nfs4_remove_reclaim_record 0
nfs_free_createdata 0
nfs_free_unlinkdata 0
n_hdlc_release 0
nouveau_abi16_ntfy_fini 1
nouveau_gem_object_del 0
nouveau_object_destroy 0
nv50_dac_destroy 0
nv50_pior_destroy 0
nv50_sor_destroy 0
nv_poweroff_work 0
ocfs2_free_refcount_tree 0
_osd_request_free 0
osst_release_request 0
ovs_vport_free 0
oz_isoc_stream_free 0
p9_idpool_destroy 0
padata_free_pd 0
pci_mmconfig_remove 0
pcxhr_chip_free 0
pcxhr_free 0
perm_destroy 0
perm_destroy 1
pkt_free_packet_data 0
pm3393_destroy 0
pmu_dev_release 0
pnp_free_resource 0
portman_free 0
pps_device_destruct 0
psb_intel_i2c_destroy 0
psb_mmu_driver_takedown 0
psb_mmu_free_pt 0
pstore_ftrace_seq_stop 1
pt1_free_adapter 0
put_nfs_open_dir_context 0
pvr2_hdw_destroy 0
pvr2_sysfs_release 0
pvr2_v4l2_destroy_no_lock 0
qib_free_fast_reg_page_list 0
qib_user_sdma_queue_destroy 0
qlcnic_sriov_cleanup_transaction 0
qlt_release 0
qp_host_free_queue 0
queue_delete 0
quickstart_button_del 0
qxl_ring_free 0
r10bio_pool_free 0
r1bio_pool_free 0
r8712_free_cmd_obj 0
r8712_getbbrfreg_cmdrsp_callback 1
r8712_readtssi_cmdrsp_callback 1
radeon_connector_destroy 0
radeon_dp_connector_destroy 0
radeon_i2c_destroy 0
radeon_vm_bo_rmv 1
```

```
range_tr_destroy 0
range_tr_destroy 1
rate_control_pid_free_sta 2
__rds_put_mr_final 0
realloc_argv 1
recent_entry_remove 1
redrat3_delete 0
regmap_del_irq_chip 1
regmap_field_free 0
regmap_mmio_free_context 0
release_memory_resource 0
__remove_pg_pool 1
__remove_xattr 1
rem_res_tree 0
rfkill_disconnect 0
rio_release_outb_dbell 1
role_destroy 0
role_destroy 1
rose_remove_node 0
rose_remove_route 0
rpc_destroy_pipe_data 0
rpc_free_iostats 0
rtd_release 0
saa7164_buffer_dealloc_user 0
sbp_free_request 0
sbp_management_agent_unregister 0
sbp_target_agent_unregister 0
sb_register_oss 0
sb_unload 0
selinux_release_secctx 0
selinux_tun_dev_free_security 0
sens_destroy 0
sens_destroy 1
slhc_free 0
smsdvb_unregister_client 0
smtc_free_fb_info 0
snd_ak4113_free 0
snd_ak4114_free 0
snd_ak4117_free 0
snd_ca0106_free 0
snd_card_do_free 0
snd_cs5530_free 0
snd_cs5535audio_free 0
snd_emu10k1_free 0
snd_hda_bus_free 0
snd_hwdep_free 0
snd_i2c_device_free 0
snd_lx6464es_free 0
snd_midi_channel_free_set 0
snd_mixart_chip_free 0
snd_mixart_free 0
snd_mts64_free 0
snd_opl3_free 0
snd_pcm_oss_release_file 0
snd_pcm_plugin_free 0
snd_sbdsp_free 0
snd_seq_device_free 0
snd_uart16550_free 0
snd_usb_audio_free 0
snd_usb_mixer_free 0
__snd_util_mem_free 1
snd_util_memhdr_free 0
snd_vx222_free 0
solo_enc_free 0
_sp2d_free 0
squashfs_cache_delete 0
```



```
src_mgr_destroy 0
st_release_request 0
stub_device_free 0
sum_mgr_destroy 0
system_root_device_release 0
t1_espi_destroy 0
t1_tp_destroy 0
target_fabric_configfs_free 0
__team_option_inst_del 0
tgfx_remove 0
tipc_free_entry 0
tomoyo_memory_free 0
tower_delete 0
tpci200_release_device 0
ttm_transfered_destroy 0
tty_audit_buf_free 0
type_destroy 0
type_destroy 1
udl_connector_destroy 0
udl_crtc_destroy 0
udl_enc_destroy 0
unifi_free_card 0
_unload_intf_hdl 0
unregister_from_lirc 0
unx_free_cred 0
usb_cleanup 0
usb_free_descriptors 0
usbhsh_ureq_free 1
usblp_cleanup 0
usb_pcwd_delete 0
user_destroy 0
user_destroy 1
v4l2_m2m_release 0
vb2_dma_contig_cleanup_ctx 0
vb2_put_vma 0
vcs_poll_data_free 0
vfio_group_unlock_and_free 0
via_aux_free 0
vic_provinfo_free 0
video_device_release 0
vmci_handle_arr_destroy 0
vme_dma_free_attribute 0
vme_dma_list_free 0
vme_lm_free 0
vme_slave_free 0
vmw_sou_destroy 0
__vxge_hw_channel_free 0
w1_slave_detach 0
wl1271_rx_filter_free 0
wlc_phy_shim_detach 0
w_resync_finished 0
xencons_free 0
xen_pcibk_config_field_free 0
xfrm_policy_destroy 0
xhci_free_container_ctx 1
xhci_segment_free 1
xt_unregister_table 0
xz_dec_lzma2_end 0
zbud_destroy_pool 0
zoran_vdev_release 0
zram_meta_free 0
zs_destroy_pool 0
```

```
**********************************************************
      82 Fri Dec 21 15:00:21 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.frees_argument.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 free_cell 0
   2 free_scq 0
   3 free_scq 1
   4 setup_card 0
   5 krealloc 0
   6 free_urbs 0
   7 free_area 0
```

```
**********************************************************
    4851 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.gfp_flags
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of GFP flag parameters.
// generated by 'gen_gfp_flags.sh'
aa_alloc_file_context 0
aa_alloc_task_context 0
aligned_kmalloc 1
alloc_buffer 1
alloc_bulk_urbs_generic 5
alloc_cmd_box 1
alloc_cpu_rmap 1
alloc_ep 1
alloc_new_range 2
__alloc_objio_seg 1
alloc_pl 0
alloc_send_rmpp_list 1
alloc_ts_config 1
alloc_workqueue_attrs 0
apparmor_cred_alloc_blank 1
apparmor_cred_prepare 2
array_zalloc 2
asd_alloc_edbs 1
asd_alloc_escbs 1
ath_rate_alloc_sta 2
b44_alloc_consistent 1
bch_bio_split 2
bio_alloc 0
bio_alloc_bioset 0
bio_alloc_drbd 0
bio_alloc_map_data 2
bio_alloc_mddev 0
bio_chain_clone 1
bio_clone 1
bio_clone_bioset 1
bio_clone_kmalloc 1
bio_clone_mddev 1
bio_clone_range 3
bio_integrity_alloc 1
bio_integrity_clone 2
bio_kmalloc 0
bio_kmalloc 1
__bio_map_kern 3
bio_map_kern 3
bl_alloc_layout_hdr 1
bl_alloc_lseg 2
blk_rq_prep_clone 3
bm_init 2
bnx2_alloc_rx_data 3
br_netpoll_enable 1
btrfs_bio_alloc 3
btrfs_bio_clone 1
btrfs_dio_bio_alloc 2
btrfs_io_bio_alloc 0
call_usermodehelper_setup 3
carl9170_usb_alloc_rx_urb 1
cas_page_alloc 1
ceph_alloc_page_vector 1
ceph_buffer_new 1
ceph_create_snap_context 1
cfg80211_inform_bss 9
compressed_bio_alloc 2
core_tmr_alloc_req 3
```



```
create_endpoint_and_queue_bulk 7
create_endpoint_and_queue_control 5
create_endpoint_and_queue_int 7
cxgbi_alloc_big_mem 1
_dev_list_add 3
devm_kzalloc 1
devres_open_group 2
drm_mm_get_block_generic 4
drm_mm_get_block_range_generic 6
drm_mm_kmalloc 1
dummy_urb_enqueue 2
dwc2_hcd_qh_create 2
dwc2_hcd_qtd_add 3
dwc2_hcd_urb_alloc 2
dwc2_hcd_urb_enqueue 3
dwc3_gadget_ep_alloc_request 1
ehci_mem_init 1
ep_alloc_request 1
ext4_kvmalloc 1
ext4_kvzalloc 1
__fa_get_part 2
fb_alloc_cmap_gfp 3
filelayout_alloc_layout_hdr 1
filelayout_alloc_lseg 2
flex_array_alloc 2
flush_entry_alloc 0
fotg210_ep_alloc_request 1
fsm_init 2
__fuse_request_alloc 1
get_1284_register 3
get_swap_bio 0
goku_alloc_request 1
gss_import_sec_context 5
gss_import_sec_context_kerberos 4
heap_init 2
hsi_alloc_controller 1
hsi_alloc_msg 1
ib_sa_guid_info_rec_query 7
ib_sa_mcmember_rec_query 7
ib_sa_path_rec_get 6
ib_sa_service_rec_query 7
ioat2_alloc_ring 2
ioat_dma_alloc_descriptor 1
iso_sched_alloc 1
iso_stream_alloc 0
itd_urb_transaction 3
kcalloc 2
kcryptd_io_read 1
kdb_strdup 1
__kmalloc 1
kmalloc 1
kmalloc_array 2
kmp_init 2
kstruprdup 1
kzalloc 1
libipw_alloc_txb 3
m66592_alloc_request 1
mca_bucket_alloc 2
mempool_kmalloc 0
minstrel_alloc_sta 2
minstrel_ht_alloc_sta 2
mpage_alloc 3
mthca_alloc_mailbox 1
mv_alloc_request 1
mv_u3d_alloc_request 1
net2272_alloc_request 1
```

```
net2280_alloc_request 1
netlbl_secattr_alloc 0
netlbl_secattr_cache_alloc 0
netlbl_secattr_catmap_alloc 0
netlbl_secattr_catmap_setbit 2
netlbl_secattr_catmap_setrng 3
new_task_smack 2
new_writequeue_entry 1
nfs4_alloc_pages 1
nfs4_alloc_state_owner 2
nfs4_find_or_create_slot 3
nfs4_label_alloc 1
nfs4_new_slot 3
nfs4_pnfs_remotestr 1
nfs_alloc_seqid 1
objio_alloc_lseg 4
objlayout_alloc_layout_hdr 1
_osd_request_alloc 0
osd_start_request 1
oz_build_endpoints_for_interface 3
oz_ep_alloc 0
pch_udc_alloc_request 1
pool_alloc_page 1
posix_acl_alloc 1
posix_acl_from_mode 1
pscsi_get_bio 0
pxa_ep_alloc_request 1
qla2x00_alloc_fcport 1
qset_add_urb 3
qset_add_urb_sg 3
qset_fill_page_list 2
qset_new_std 3
r10bio_pool_alloc 0
r10buf_pool_alloc 0
r1bio_pool_alloc 0
r1buf_pool_alloc 0
r8a66597_alloc_request 1
rate_control_pid_alloc_sta 2
rcu_string_strdup 1
rds_ib_conn_alloc 1
rds_iw_conn_alloc 1
rds_loop_conn_alloc 1
rds_message_alloc 1
request_firmware_nowait 4
resize_iovec 1
rfcomm_dlc_alloc 0
rpipe_get_idle 3
rtl_rate_alloc_sta 2
rx_alloc_submit 1
sctp_add_bind_addr 3
sctp_association_new 3
sctp_auth_asoc_set_secret 3
sctp_auth_create_key 1
sctp_auth_shkey_create 1
sctp_bind_addr_copy 4
sctp_bind_addr_dup 2
sctp_bind_addrs_to_raw 2
sctp_copy_one_addr 4
sctp_datamsg_new 0
sctp_endpoint_new 1
sctp_make_init_ack 2
sctp_transport_new 2
sctp_tsnmap_init 3
sdebug_device_create 1
sdev_evt_alloc 1
sdev_evt_send_simple 2
```



```
security_context_to_sid_core 4
security_context_to_sid_default 4
selinux_cred_alloc_blank 1
selinux_sk_alloc_security 2
set_1284_register 3
sg_kmalloc 1
sierra_setup_urb 5
sitd_urb_transaction 3
smack_cred_alloc_blank 1
smack_cred_prepare 2
srp_alloc_iu 2
sta_info_alloc 2
submit_async_request 5
target_submit_tmr 6
team_port_enable_netpoll 2
ubi_zalloc_vid_hdr 1
udc_alloc_request 1
ulist_alloc 0
update_display_visual 1
usb_alloc_urb 1
usbhsg_ep_alloc_request 1
usbhsh_data_stage_packet_push 3
usbhsh_endpoint_attach 2
usbhsh_queue_push 2
usbhsh_status_stage_packet_push 3
usbhsh_ureq_alloc 2
vic_provinfo_alloc 0
xfrm_policy_alloc 1
xhci_alloc_command 3
xhci_alloc_container_ctx 2
xhci_alloc_segments_for_ring 6
xhci_alloc_tt_info 4
xhci_ring_alloc 4
xhci_ring_expansion 3
xhci_segment_alloc 2
xpc_kmalloc_cacheline_aligned 1
xpc_kzalloc_cacheline_aligned 1
zbud_create_pool 0
```

```
**********************************************************
     110 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.ignore_macro_indenting
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 while_each_pid_thread
   2 while_each_pid_task
   3 LOCKDEP_STATE
   4 get_user_catch
   5 put_user_catch
   6 tcp_skb_tsorted_restore
```

```
**********************************************************
      74 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.ignore_memcpy_struct_overflows
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 /* list for struct types to ignore */
   2 lpfc_fdmi_attr_entry
   3 be_cmd_req_hdr
```

```
**********************************************************
    1146 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.ignore_side_effects
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
/*
 * Manually created.
 *
 * Most of these have intentional side effects.
 * Some of them like put_user() and friends, have side effects when __CHECKER__
 * is defined but not in the compiled kernel.
 */
ADD_STA_STATS
ARCH_DLINFO
AWDATA
ENCODE
ENCODE_DATA
ENCODE_STR
get_child
get_child_rcu
get_unaligned
get_user
__get_user
__get_user_nocheck
hybrid_tuner_request_state
iterate_bvec
iterate_all_kinds
lookup
lookup_rightempty
MAKE_RAW_BYTE
MAKE_RAW_BYTE_56K
mdelay
MsgHead
MUL64
NEW_AUX_ENT
nh_vmac_nhbytes
ntohl
OUT_RING_REG
poly_step
PUT_BYTE
put_short
put_user
__put_user
__put_user_nocheck
R128_WAIT_UNTIL_PAGE_FLIPPED
R600_CLEAR_AGE
R600_DISPATCH_AGE
R600_FRAME_AGE
RADEON_CLEAR_AGE
RADEON_DISPATCH_AGE
RADEON_FLUSH_CACHE
RADEON_FRAME_AGE
RADEON_PURGE_CACHE
RADEON_PURGE_ZCACHE
RADEON_WAIT_UNTIL_2D_IDLE
RADEON_WAIT_UNTIL_3D_IDLE
RADEON_WAIT_UNTIL_IDLE
RCU_INIT_POINTER
READ64
rtnl_dereference
send_bits
send_code
SOCK_ADDR_LOAD_NESTED_FIELD
SOCK_ADDR_LOAD_NESTED_FIELD_SIZE_OFF
SOCK_ADDR_LOAD_OR_STORE_NESTED_FIELD_SIZE_OFF
```



```
SOCK_ADDR_LOAD_OR_STORE_NESTED_FIELD
SOCK_OPS_GET_TCP32
unsafe_get_user
unsafe_put_user
VIA_OUT_RING_QW
WRITE64
Z
```

```
**********************************************************
    2072 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.ignore_uninitialized_param
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 regmap_read 2
  2 regmap_fields_read 2
  3 visorchannel_read 2
  4 dmam_alloc_coherent 2
  5 diva_pci_alloc_consistent 2
  6 read_mos_reg 3
  7 adp5520_read 2
  8 gameport_cooked_read 2
  9 max3100_sr 1
 10 sata_scr_read 2
 11 svia_scr_read 2
 12 lp8788_read_byte 2
 13 qla83xx_rd_reg 2
 14 cciss_read_capacity 2
 15 cciss_read_capacity 3
 16 rio_mport_read_config_32 4
 17 acpi_read 0
 18 axi_clkgen_mmcm_read 2
 19 intel_msic_irq_read 2
 20 pci_user_read_config_word 2
 21 ec_read 1
 22 sony_call_snc_handle 2
 23 pci_user_read_config_word 2
 24 read_reg_fp 2
 25 vid_blk_read_word 2
 26 mc417_memory_read 2
 27 stv06xx_read_sensor 2
 28 lm90_read_reg 2
 29 read_mii_word 3
 30 read_eprom_word 2
 31 generic_ocp_read 2
 32 lan78xx_read_reg 2
 33 com20020_copy_from_card 3
 34 wl3501_get_from_wla 2
 35 ipw_get_ordinal 2
 36 generic_ocp_read 3
 37 et131x_mii_read 2
 38 ql_mii_read_reg 2
 39 atl1c_read_phy_dbg 2
 40 atl2_read_phy_reg 2
 41 atl1_read_phy_reg 2
 42 pch_gbe_hal_read_phy_reg 2
 43 t1_tpi_read 2
 44 rio_local_read_config_32 2
 45 acpi_smbus_read 4
 46 pci_read_config_dword 2
 47 viafb_i2c_readbyte 3
 48 bap_read 1
 49 of_get_property 2
 50 of_property_read_u32 2
 51 of_property_read_u8 2
 52 of_property_read_u16 2
 53 of_property_read_u32_index 3
 54 intel_gvt_hypervisor_read_gpa 2
 55 cs5536_read 1
 56 __amd64_read_pci_cfg_dword 2
 57 e1e_rphy 2
 58 imx_phy_reg_read 0
 59 chipio_read 0
 60 had_read_register 1
```



```
 61 qcaspi_read_register 2
 62 mv88e6xxx_g2_read 2
 63 b53_read8 3
 64 b53_read16 3
 65 b53_read32 3
 66 b53_read48 3
 67 b53_read64 3
 68 dvbtqam_get_acc_pkt_err 1
 69 ch7xxx_readb 2
 70 ivch_read 2
 71 tvp7002_read 2
 72 rtsx_pci_read_register 2
 73 rtsx_usb_ep0_read_register 2
 74 __t1_tpi_read 2
 75 smsc95xx_read_reg 2
 76 pci_user_read_config_dword 2
 77 da903x_read 2
 78 rio_read_config_8 2
 79 rio_read_config_16 2
 80 rio_read_config_32 2
 81 __ad7280_read32 1
 82 rtsx_read_cfg_dw 3
 83 lola_read_param 3
 84 soc_dapm_read 2
 85 read_nic_byte 2
 86 amd_smn_read 2
 87 meson_ao_cec_read 2
 88 pci_read_config_byte 2
 89 pci_read_config_word 2
 90 i40e_read_nvm_word 2
 91 stk_camera_read_reg 2
 92 cnl_get_buf_trans_edp 1
 93 cnl_get_buf_trans_dp 1
 94 intel_ddi_get_buf_trans_edp 1
 95 intel_ddi_get_buf_trans_dp 1
 96 cnl_get_buf_trans_hdmi 1
 97 iosf_mbi_read 3
 98 lola_codec_read 5
 99 chipio_read 2
100 pcxhr_write_io_num_reg_cont 3
101 read_current_timer 0
102 pwrap_read 2
103 dibusb_read_eeprom_byte 2
104 of_fdt_unflatten_tree 2
```

**********
319 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.ignored_macros
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
/*
 * These macros are actively misleading to smatch so
 * it's best to ignore them.
 */
WARN_ON
B43legacy_WARN_ON
B43_WARN_ON
/* ASSERT can be worthwhile or bogus. */
HPI_DEBUG_ASSERT
snd_BUG_ON
WARN_ON_ONCE
ubi_assert
bfa_assert
ZD_ASSERT
ubifs_assert
AR_DEBUG_ASSERT
RT_ASSERT
SKD_ASSERT
SNIC_BUG_ON
LASSERT
AA_BUG
```

```
************************************************************
    3604 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.ioctls
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
************************************************************
```

```
aac_cfg_ioctl
ac_ioctl
acq_ioctl
adpt_unlocked_ioctl
adsp_ioctl
advwdt_ioctl
adx_wdt_ioctl
agp_ioctl
ali_ioctl
anslcd_ioctl
apm_ioctl
ar7_wdt_ioctl
asr_ioctl
at32_wdt_ioctl
at91_wdt_ioctl
audamrnb_ioctl
audevrc_ioctl
audio_in_ioctl
audio_ioctl
audpp_ioctl
audpre_ioctl
audqcelp_ioctl
autofs_root_ioctl
bcm47xx_wdt_ioctl
bcm_char_ioctl
bfin_otp_ioctl
bfin_wdt_ioctl
booke_wdt_ioctl
bridge_ioctl
capi_unlocked_ioctl
chd_dec_ioctl
ch_ioctl
chsc_ioctl
cmm_ioctl
coh901327_ioctl
comedi_unlocked_ioctl
cosa_chardev_ioctl
cpu5wdt_ioctl
cpwd_ioctl
_ctl_ioctl
ctrl_cdev_ioctl
d7s_ioctl
dabusb_ioctl
davinci_wdt_ioctl
dev_ioctl
dm_ctl_ioctl
drm_ioctl
ds1620_unlocked_ioctl
dsp56k_ioctl
dst_ca_ioctl
dtlk_ioctl
dv1394_ioctl
dvb_ca_en50221_io_ioctl
dvb_demux_ioctl
dvb_dvr_ioctl
dvb_generic_ioctl
dvb_net_ioctl
easycap_ioctl
easysnd_ioctl
efi_rtc_ioctl
```



```
envctrl_ioctl
ep93xx_wdt_ioctl
ep_ioctl
epx_c3_ioctl
erst_dbg_ioctl
esb_ioctl
eurwdt_ioctl
evdev_ioctl
evtchn_ioctl
fb_ioctl
ffs_ep0_ioctl
ffs_epfile_ioctl
fitpc2_wdt_ioctl
flash_ioctl
fop_ioctl
fs3270_ioctlioctl
ft1000_ChIoctl
fw_device_op_ioctl
gdth_unlocked_ioctl
gef_wdt_ioctl
gen_rtc_unlocked_ioctl
geodewdt_ioctl
GLOB_SBD_unlocked_ioctl
gru_file_unlocked_ioctl
hiddev_ioctl
hidraw_ioctl
hpet_ioctl
hp_sdc_rtc_unlocked_ioctl
hpwdt_ioctl
hub_ioctl
hung_up_tty_ioctl
i2cdev_ioctl
i2o_cfg_ioctl
i810_ioctl
i830_ioctl
i8k_ioctl
ib_umad_ioctl
ibwdt_ioctl
idetape_chrdev_ioctl
imx2_wdt_ioctl
indydog_ioctl
intel_sst_ioctl
ioctl
ioctl_rio
iop_wdt_ioctl
iowarrior_ioctl
ip2_ipl_ioctl
ipmi_unlocked_ioctl
ir_lirc_ioctl
isdn_divert_ioctl
isdn_unlocked_ioctl
it8712f_wdt_ioctl
iTCO_wdt_ioctl
ivtv_v4l2_ioctl
ixj_ioctl
ixp2000_wdt_ioctl
ixp4xx_wdt_ioctl
joydev_ioctl
jsf_ioctl
ks8695_wdt_ioctl
lcd_ioctl
lirc_dev_fop_ioctl
lirc_ioctl
lp_ioctl
macvtap_ioctl
max63xx_wdt_ioctl
```

```
megadev_unlocked_ioctl
megasas_mgmt_ioctl
memrar_ioctl
mISDN_ioctl
mixcomwd_ioctl
mmtimer_ioctl
mon_bin_ioctl
mon_stat_ioctl
mpc8xxx_wdt_ioctl
mpcore_wdt_ioctl
mptctl_ioctl
mraid_mm_unlocked_ioctl
msm_ioctl_config
msm_ioctl_control
msm_ioctl_frame
mtd_unlocked_ioctl
mtx1_wdt_ioctl
mv64x60_wdt_ioctl
mwave_ioctl
nosy_ioctl
nuc900_wdt_ioctl
nvram_ioctl
nvram_unlocked_ioctl
octeon_wdt_ioctl
omap_wdt_ioctl
openprom_ioctl
orion_wdt_ioctl
osd_uld_ioctl
osst_ioctl
pc87413_ioctl
pcipcwd_ioctl
pcwd_ioctl
phantom_ioctl
pikawdt_ioctl
pkt_ctl_ioctl
pmcraid_chr_ioctl
pmem_ioctl
pmu_unlocked_ioctl
pnx4008_wdt_ioctl
pnx833x_wdt_ioctl
pp_ioctl
ppp_ioctl
pps_cdev_ioctl
printer_ioctl
proc_bus_pci_ioctl
pt_ioctl
random_ioctl
raw1394_ioctl
raw_ctl_ioctl
raw_ioctl
rc32434_wdt_ioctl
rdc321x_wdt_ioctl
rio_fw_ioctl
riowd_ioctl
rtc_dev_ioctl
rtc_ioctl
s3c2410wdt_ioctl
sa1100dog_ioctl
sbwdog_ioctl
sc1200wdt_ioctl
sch311x_wdt_ioctl
scx200_wdt_ioctl
sg_unlocked_ioctl
sh_wdt_ioctl
smb_ioctl
snd_ioctl
```



```
softdog_ioctl
sonypi_misc_ioctl
sp805_wdt_ioctl
spidev_ioctl
st_ioctl
stli_memioctl
stl_memioctl
stmp3xxx_wdt_ioctl
sx_fw_ioctl
tapechar_ioctl
tosh_ioctl
ts72xx_wdt_ioctl
tty_ioctl
tun_chr_ioctl
twa_chrdev_ioctl
tw_chrdev_ioctl
twl4030_wdt_ioctl
twl_chrdev_ioctl
txx9wdt_ioctl
ubi_cdev_ioctl
uctrl_ioctl
uinput_ioctl
usbdev_ioctl
usblp_ioctl
usb_pcwd_ioctl
usbtest_ioctl
usbtmc_ioctl
uv_mmtimer_ioctl
v4l2_ioctl
vfd_ioctl
vhost_net_ioctl
video1394_ioctl
video_ioctl2
video_ioctl2V4L2ioctlhandler
vino_ioctl
viotap_unlocked_ioctl
vmcp_ioctl
vme_user_unlocked_ioctl
vmwdt_ioctl
vmw_unlocked_ioctl
vol_cdev_ioctl
wafwdt_ioctl
watchdog_ioctl
wb_smsc_wdt_ioctl
wdrtas_ioctl
wdt977_ioctl
wdt_ioctl
wdtpci_ioctl
wdt_unlocked_ioctl
wm831x_wdt_ioctl
wm8350_wdt_ioctl
zcrypt_unlocked_ioctl
zfcp_cfdc_dev_ioctl
zf_ioctl
```

```
**********************************************************
      73 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.macro_takes_sizeof_argument
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 BNX2FC_RQ_BUF_LOG_SZ
   2 BNX2FC_NUM_MAX_SESS_LOG
   3 ilog2
   4 FP_XSTATE_MAGIC2_SIZE
```

```
**********************************************************
      47 Fri Dec 21 15:00:22 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.must_check_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 ERR_PTR
   2 PTR_ERR
   3 IS_ERR
   4 IS_ERR_OR_NULL
   5 ERR_CAST
```

```
**********
     180 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.no_inline_functions
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********
   1 __fswab16
   2 __fswab32
   3 __fswab64
   4 __builtin_bswap16
   5 __builtin_bswap32
   6 __builtin_bswap64
   7 __arch_hweight8
   8 __arch_hweight16
   9 __arch_hweight32
  10 __arch_hweight64
  11 __write_once_size
  12 atomic_set
```

```
**********************************************************
    1849 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.no_return_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of functions which don't return.
// generated by 'gen_no_return_funcs.sh'
__assert_fail
exit
__builtin_unreachable
ahd_handle_hwerrint
aic7xxx_panic_abort
assfail
ast_gem_init_object
audit_tree_handle_event
backref_tree_panic
bnep_session
cafe_nand_bug
cifs_demultiplex_thread
cirrus_gem_init_object
cmtp_session
complete_and_exit
cpu_bringup_and_idle
cpu_idle_loop
cpu_startup_entry
cryptomgr_probe
cryptomgr_test
denali_ecc_calculate
denali_ecc_correct
denali_ecc_hwctl
df_debug
die_kmmio_nesting_error
do_double_fault
do_exit
do_group_exit
drm_framebuffer_free_bug
drm_gem_object_ref_bug
dummy_handler
early_panic
extent_io_tree_panic
fsnotify_mark_destroy
gru_abort
hidp_session_thread
hlt_play_dead
hugetlbfs_write_end
hugetlb_vm_op_fault
i915_gem_init_object
i915_gem_object_get_pages_stolen
__invalid_creds
jffs2_garbage_collect_thread
kdb_reboot
kgdboc_reset_disconnect
kthread
kthreadd
kvm_spurious_fault
logfs_mark_segment_bad
machine_real_restart
mgag200_gem_init_object
mpt_halt_firmware
native_machine_emergency_restart
native_machine_halt
netlink_data_ready
nfs4_run_state_manager
nfsd
nfs_proc_commit_rpc_prepare
```



```
nfs_proc_commit_setup
__ocfs2_recovery_thread
ordered_data_tree_panic
panic
panic_smp_self_stop
ping_hash
prio_changed_idle
prio_changed_stop
r100_semaphore_ring_emit
r8712_cmd_thread
radeon_gem_object_init
rcu_gp_kthread
rcu_nocb_kthread
__reiserfs_panic
report_instruction_timeout
rest_init
retu_power_off
skb_over_panic
skb_panic
skb_under_panic
__stack_chk_fail
start_kernel
start_secondary
stop_self
stop_this_cpu
svc_udp_accept
switched_to_idle
switched_to_stop
SYSC_exit
SYSC_exit_group
SyS_exit
SyS_exit_group
task_tick_rt
ttm_bo_ref_bug
txd_chain
udl_gem_init_object
usbatm_do_heavy_init
vdump_bucket_and_panic
wait_for_helper
wait_for_panic
x86_64_start_kernel
x86_64_start_reservations
yield_task_stop
```

```
**********************************************************
       9 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.no_return_funcs.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 watchdog
```

**********

*560 Fri Dec 21 15:00:23 2018*
*new/usr/src/tools/smatch/src/smatch_data/kernel.parameter_implications.add*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * It's pretty common that the return value implies something about a parameter.
 * This gives you a way to tell Smatch about it.
 *
 * The format is:
 * function "return range" parameter "implied value range list of parameter"
 *
 * You have to specify at least two return ranges for a function.  The implied
 * value is a range list, but the return range is just a range and we only care
 * about the min and the max.  We start counting the first parameter at zero.
 *
 */

rw_verify_area "0-1000000" 3 "0-1000000"
rw_verify_area "-4095-(-1)" 3 "min-max"
```

```
**********************************************************
     269 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.puts_argument
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 // list of functions and the argument they decrement the ref of.
   2 // generated by 'gen_puts_list.sh'
   3 batadv_sysfs_del_hardif 0
   4 class_compat_unregister 0
   5 cleanup_glue_dir 1
   6 hugepage_exit_sysfs 0
   7 hugepage_init_sysfs 0
   8 ib_device_unregister_sysfs 0
   9 uio_dev_del_attributes 0
```

```
**********************************************************
   29413 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.returns_err_ptr
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of functions that return a new allocation.
// generated by 'gen_err_ptr_list.sh'
aa_dfa_unpack
aa_simple_write_to_buffer
aa_unpack
acer_rfkill_register
acquire_group
ad714x_probe
ad7606_probe
ad7879_probe
add_inode
add_numbered_child
add_partition
add_qgroup_rb
addrconf_add_dev
addrconf_dst_alloc
add_tracepoint
add_volume
adfs_lookup
ads7846_probe_dt
adxl34x_probe
aead_geniv_alloc
affs_alloc_extblock
affs_bread_ino
affs_find_entry
affs_getemptyblk_ino
affs_get_extblock_slow
affs_getzeroblk_ino
affs_iget
afs_cell_alloc
afs_cell_create
afs_cell_lookup
afs_dir_get_page
afs_iget
afs_iget_autocell
afs_lookup
afs_lookup_server
afs_mntpt_do_automount
afs_mntpt_lookup
afs_mount
afs_try_auto_mntpt
afs_vlocation_lookup
afs_volume_lookup
afs_volume_pick_fileserver
alg_get_type
alloc
allocate_fake_cpuc
allocate_partition
alloc_cmd
alloc_cmd_box
alloc_counters
alloc_huge_page
alloc_log_tree
alloc_mnt_ns
alloc_msg
alloc_nfs_open_context
alloc_nfs_open_dir_context
alloc_param_target
alloc_rbio
alloc_sdesc
```



```
alloc_swap_info
alloc_trace_uprobe
alloc_ts_config
alloc_upcall
alloc_vmap_area
anon_inode_getfile
anon_inode_mkinode
applesmc_get_entry_by_index
applesmc_get_entry_by_key
arpt_register_table
ast_user_framebuffer_create
ath10k_fetch_fw_file
ath10k_htc_create
ath5k_txq_setup
ath6kl_cfg80211_add_iface
atk_ggrp
atk_gitm
atk_sitm
audit_data_to_entry
audit_dupe_rule
audit_dupe_watch
audit_init_parent
audit_init_watch
audit_to_entry_common
audit_unpack_string
auo_pixcir_parse_dt
autofs4_d_automount
autofs4_lookup
b43_wireless_init
backlight_device_register
bad_inode_lookup
balloon_devinfo_alloc
balloon_mapping_alloc
bar_init
bch_btree_node_alloc
bch_btree_node_get
bclean
bcma_hcd_create_pdev
bd_start_claiming
befs_iget
befs_lookup
beiscsi_ep_connect
bfs_iget
bfs_lookup
bio_copy_user_iov
__bio_map_kern
bio_map_kern
__bio_map_user_iov
bl_alloc_lseg
bl_find_get_zeroing_page
blkcg_css_alloc
blkdev_get_by_dev
blkdev_get_by_path
blkg_create
blkg_lookup_create
blk_make_request
bnx2i_ep_connect
brcmf_add_if
brcmf_alloc_vif
brcmf_bus_gettxq
brcmf_cfg80211_add_iface
brcmf_fws_macdesc_lookup
brcmf_p2p_add_vif
brcmf_p2p_create_p2pdev
brcmf_setup_wiphy
brd_probe
```

```
br_multicast_get_group
br_multicast_new_group
bsg_add_device
bsg_alloc_command
bsg_get_device
bsg_get_done_cmd
bsg_map_hdr
btree_get_extent
btrfs_alloc_free_block
btrfs_create_tree
btrfs_get_acl
btrfs_get_dentry
btrfs_get_extent
btrfs_get_extent_fiemap
btrfs_get_or_create_delayed_node
btrfs_get_parent
btrfs_iget
btrfs_init_new_buffer
btrfs_lookup_csum
btrfs_lookup_dentry
btrfs_lookup_dir_index_item
btrfs_lookup_dir_item
btrfs_lookup_inode_extref
btrfs_lookup_inode_ref
btrfs_lookup_xattr
btrfs_mount
btrfs_new_extent_direct
btrfs_new_inode
btrfs_reada_add
btrfs_read_fs_root_no_name
btrfs_read_tree_root
btrfs_ref_to_path
btrfs_search_dir_index_item
bu21013_parse_dt
build_backref_tree
build_map_info
build_unc_path_to_root
c2_ah_create
c2_alloc_pd
c2_alloc_ucontext
c2_create_cq
c2_create_qp
c2port_device_register
c2_reg_phys_mr
c2_reg_user_mr
c4iw_ah_create
c4iw_alloc
c4iw_allocate_pd
c4iw_alloc_fast_reg_mr
c4iw_alloc_fastreg_pbl
c4iw_alloc_mw
c4iw_alloc_ucontext
c4iw_create_cq
c4iw_create_qp
c4iw_get_dma_mr
c4iw_register_phys_mem
c4iw_reg_user_mr
c4iw_uld_add
cache_create_net
cachefiles_alloc_object
cachefiles_check_active
cachefiles_get_directory
carl9170_alloc
cciss_seq_start
ceph_alloc_page_vector
ceph_auth_init
```



```
ceph_create_client
ceph_create_snap_realm
ceph_extract_encoded_string
ceph_fh_to_parent
ceph_finish_lookup
ceph_get_direct_page_vector
ceph_get_inode
ceph_lookup
ceph_mdsc_build_path
ceph_mdsc_create_request
ceph_mdsmap_decode
ceph_monmap_decode
ceph_mount
ceph_osdc_new_request
ceph_parse_options
ceph_real_mount
cfg80211_get_dev_from_ifindex
__cfg80211_rdev_from_attrs
__cfg80211_wdev_from_attrs
__cfh_to_dentry
cgroup_css_from_dir
cgroup_lookup
cgroup_mount
cgroup_root_from_opts
cgrp_css_alloc
chainiv_alloc
check_partition
check_range
cifs_build_devname
cifs_compose_mount_options
cifs_construct_tcon
cifs_dfs_do_automount
cifs_do_mount
cifs_get_root
cifs_get_smb_ses
cifs_get_spnego_key
cifs_get_tcon
cifs_get_tcp_session
cifs_get_volume_info
cifs_lookup
cifs_root_iget
cifs_sb_tlink
cifs_setup_async_request
cifs_setup_request
ci_hdrc_add_device
cirrus_user_framebuffer_create
__class_create
__clk_register
clk_register
clk_register_composite
clk_register_fixed_factor
clk_register_fixed_rate
clk_register_gate
clk_register_mux_table
clone_fs_devices
clone_mnt
clone_uts_ns
clusterip_seq_start
cma3000_init
cm_copy_private_data
cm_create_timewait_info
coda_cnode_make
coda_cnode_makectl
coda_iget
coda_lookup
config_desc_make
```

```
configfs_lookup
configfs_new_dirent
construct_key_and_link
context_alloc
copy_dev_ioctl
copy_gadget_strings
copy_msg
copy_net_ns
copy_pid_ns
copy_process
copy_tree
copy_znode
core_alloc_hba
core_alloc_port
core_alua_allocate_lu_gp
core_alua_allocate_lu_gp_mem
core_alua_allocate_tg_pt_gp_mem
core_dev_add_lun
core_tpg_add_initiator_node_acl
core_tpg_pre_addlun
core_tpg_pre_dellun
cpuacct_css_alloc
cpu_cgroup_css_alloc
cpufreq_cooling_register
cpuset_css_alloc
cpuset_mount
create_and_register_pcpu
create_empty_lvol
create_entry
create_fs_client
create_hw_context
create_iboe_ah
create_ipc_ns
create_new_namespaces
create_pid_namespace
create_pinned_em
create_reloc_inode
create_request_message
_create_sg_bios
create_xattr_datum
create_xattr_ref
crush_decode
cryptd_alloc_ablkcipher
cryptd_alloc_aead
cryptd_alloc_ahash
cryptd_alloc_instance
crypto_alg_mod_lookup
crypto_alloc_ablkcipher
crypto_alloc_aead
crypto_alloc_base
crypto_alloc_instance
crypto_alloc_instance2
__crypto_alloc_tfm
crypto_alloc_tfm
crypto_attr_alg_name
crypto_authenc_ahash
crypto_authenc_ahash_fb
crypto_authenc_alloc
crypto_authenc_esn_ahash
crypto_authenc_esn_alloc
crypto_cbc_alloc
crypto_ccm_alloc
crypto_ccm_alloc_common
crypto_ccm_base_alloc
crypto_create_tfm
crypto_ctr_alloc
```



```
crypto_cts_alloc
crypto_ecb_alloc
crypto_fpu_alloc
crypto_gcm_alloc
crypto_gcm_alloc_common
crypto_gcm_base_alloc
crypto_get_attr_type
crypto_larval_alloc
crypto_larval_lookup
crypto_larval_wait
crypto_lookup_aead
crypto_lookup_skcipher
crypto_pcbc_alloc
crypto_rfc3686_alloc
crypto_rfc4106_alloc
crypto_rfc4309_alloc
crypto_rfc4543_alloc
crypto_spawn_alg
crypto_spawn_tfm
crypto_user_aead_alg
crypto_user_skcipher_alg
ctnetlink_create_conntrack
ctrl_build_family_msg
ctrl_build_mcgrp_msg
cxgbi_check_route
cxgbi_ep_connect
cyttsp4_probe
cyttsp_probe
d_absolute_path
d_add_ci
db9_probe
debug_css_alloc
dentry_open
__dentry_path
dentry_path
devcgroup_css_alloc
devfreq_add_device
device_create_vargs
devm_backlight_device_register
devm_clk_get
devm_clk_register
devm_ioremap_resource
devm_lcd_device_register
devm_of_pwm_get
devm_pinctrl_get_select
devm_pwm_get
devm_regmap_field_alloc
devm_regmap_init
devm_regulator_get
devm_reset_control_get
devm_rtc_device_register
devm_usb_get_phy_by_phandle
devpts_mount
devpts_pty_new
d_hash_and_lookup
dio_get_page
dirty_cow_bottom_up
dirty_cow_nnode
dirty_cow_pnode
dirty_cow_znode
disk_seqf_start
dlm_register_domain
dl_seq_start
dma_buf_attach
dma_buf_export_named
dma_buf_get
```

```
dma_buf_map_attachment
d_materialise_unique
dm_block_manager_create
dm_bufio_client_create
dm_build_path_uevent
dm_cache_metadata_open
dm_hash_rename
dm_io_client_create
dm_kcopyd_client_create
dm_pool_metadata_open
dm_region_hash_create
dm_sm_disk_create
dm_sm_disk_open
dm_sm_metadata_init
dm_swap_table
dm_tm_create
do_add_page_to_bio
d_obtain_alias
doc_probe_device
do_create
do_file_open_root
do_find_free_space
do_open
do_read_cache_page
__d_path
d_path
drbd_nla_find_nested
drbd_request_prepare
drm_gem_map_dma_buf
drm_gem_prime_import
drm_prime_pages_to_sg
drm_sysfs_create
ds620_update_client
dsa_switch_setup
dummy_allocate_iso_context
__d_unalias
dwc3_alloc_one_event_buffer
dw_dma_cyclic_prep
dynamic_dname
e100_request_firmware
ebt_register_table
ecm_alloc
ecm_alloc_inst
ecryptfs_do_create
__ecryptfs_get_inode
ecryptfs_lookup
ecryptfs_mount
edid_load
eem_alloc
eem_alloc_inst
efivarfs_alloc_dentry
efivarfs_lookup
efs_get_parent
efs_iget
efs_nfs_get_inode
emergency_read_begin
enclosure_component_register
enclosure_register
encrypted_key_alloc
eseqiv_alloc
eventfd_ctx_fileget
eventfd_fget
eventfd_file_create
exofs_get_page
exofs_get_parent
exofs_iget
```



```
exofs_lookup
exofs_mount
exofs_new_inode
exofs_nfs_get_inode
exp_find_key
exp_get_by_name
exportfs_decode_fh
ext2_acl_from_disk
ext2_acl_to_disk
ext2_get_acl
ext2_get_inode
ext2_get_page
ext2_get_parent
ext2_iget
ext2_lookup
ext2_new_inode
ext2_nfs_get_inode
ext3_acl_from_disk
ext3_acl_to_disk
ext3_get_acl
ext3_get_parent
ext3_iget
ext3_journal_start_sb
ext3_lookup
ext3_new_inode
ext3_nfs_get_inode
ext3_orphan_get
ext4_acl_from_disk
ext4_acl_to_disk
ext4_append
ext4_ext_find_extent
ext4_get_acl
ext4_get_parent
ext4_iget
__ext4_journal_start_reserved
__ext4_journal_start_sb
ext4_lookup
__ext4_new_inode
ext4_nfs_get_inode
ext4_orphan_get
__ext4_read_dirblock
extract_hostname
extract_sharename
f2fs_acl_from_disk
f2fs_acl_to_disk
f2fs_get_acl
f2fs_get_parent
f2fs_iget
f2fs_lookup
f2fs_new_inode
f2fs_nfs_get_inode
fanotify_add_new_mark
fanotify_merge
fat_build_inode
fault_create_debugfs_attr
fcoe_if_create
fcoe_interface_create
ff_dev_create
__fh_to_dentry
fib6_add_1
fib_create_info
fib_rules_register
__file_cft
file_open_root
find_alloc_undo
find_data_page
```

```
find_devfreq_governor
find_device_devfreq
find_get_context
find_hosted_bus
find_keyring_by_name
find_lively_task_by_vpid
find_msg
find_nfs_version
find_subdir
__flow_tbl_rehash
__fmc_scan_sdb_tree
follow_huge_addr
follow_page_mask
follow_trans_huge_pmd
freeze_bdev
freezer_css_alloc
freq_reg_info_regd
frontswap_register_ops
__fscache_lookup_cache_tag
fsm_init
fsnotify_add_notify_event
fsnotify_alloc_group
fs_set_subtype
ft_add_acl
function_make
fuse_d_add_directory
fuse_get_dentry
fuse_get_parent
__fuse_get_req
fuse_lookup
fw_create_instance
fwserial_claim_port
g762_update_client
gadget_config_name_strings_make
gadgets_make
gadget_strings_strings_make
gc_probe
generic_create_cred
get_authorizer
get_buffer
get_cifs_acl_by_fid
get_cifs_acl_by_path
get_connect_authorizer
get_cramfs_inode
get_default_root
__get_device_from_cb
get_device_pmkids
get_domain
get_empty_filp
get_fb_info
get_free_i2c_dev
geth_alloc
geth_alloc_inst
gether_connect
gether_setup_name
gether_setup_name_default
get_highmem_page_buffer
get_i2c
get_key
get_ldops
get_lock_data_page
get_mtd_device
get_mtd_device_nm
getname_flags
get_net_ns_by_fd
get_net_ns_by_pid
```



```
get_new_cssid
get_new_data_page
get_node_page
get_node_page_ra
get_one_event
__get_or_create_frag
get_phy_device
__get_policy_once
_get_posix_acl
get_raw_socket
get_socket
get_subdir
get_sysctl
get_tap_socket
get_ticket_handler
__get_txreq
get_user_arg_ptr
get_vfsmount_from_fd
get_vlan
gfs2_dirent_scan
gfs2_dirent_search
gfs2_dir_get_hash_table
gfs2_dir_search
gfs2_get_acl
gfs2_get_dentry
gfs2_inode_lookup
__gfs2_lookup
gfs2_lookup_by_inum
gfs2_lookupi
gfs2_lookup_simple
gfs2_mount
gfs2_mount_meta
ghes_new
gpio_keys_get_devtree_pdata
gpio_leds_create_of
gre_create
gre_gso_segment
gre_handle_offloads
gru_alloc_gts
gru_alloc_locked_gts
gru_register_mmu_notifier
gss_alloc_msg
gss_create
gss_create_cred
gss_fill_context
gss_setup_upcall
halve
hbcall_from_type
__hci_cmd_sync_ev
hci_connect
hci_connect_acl
hci_connect_le
hci_connect_sco
hci_get_cmd_complete
hdlcdrv_register
hfs_bmap_alloc
hfs_bnode_create
hfs_bnode_find
hfs_bnode_split
hfs_file_lookup
hfs_lookup
hfsplus_bmap_alloc
hfsplus_bnode_create
hfsplus_bnode_find
hfsplus_file_lookup
hfsplus_iget
```

```
hfsplus_lookup
hid_allocate_device
hpfs_lookup
hugetlb_cgroup_css_alloc
hugetlb_file_setup
hvc_alloc
hwmon_device_register
i2400m_get_device_info
i2400m_msg_to_dev
i2o_block_device_alloc
i2o_block_request_alloc
i2o_device_alloc
i2o_iop_alloc
i2o_msg_get
i2o_msg_get_wait
i2o_scsi_host_alloc
i915_gem_context_get_hang_stats
i915_gem_dmabuf_vmap
i915_gem_map_dma_buf
i915_gem_prime_import
ialloc
ib_alloc_fast_reg_mr
ib_alloc_fast_reg_page_list
ib_alloc_fmr
ib_alloc_mw
ib_alloc_xrcd
ib_create_ah_from_wc
ib_create_cm_id
ib_create_fmr_pool
ib_create_send_mad
ib_create_srq
ib_fmr_pool_map_phys
__ib_open_qp
ib_open_qp
ib_redirect_mad_qp
ib_register_mad_agent
ib_register_mad_snoop
ib_reg_phys_mr
ib_sa_join_multicast
ib_ucm_ctx_get
ib_umem_get
ib_uverbs_alloc_event_file
ib_uverbs_unmarshall_recv
icmp6_dst_alloc
icmp_route_lookup
icmpv6_route_lookup
idle_thread_get
id_map_alloc
idr_replace
ieee80211_add_iface
ieee80211_key_alloc
ieee80211_new_chanctx
iget_xattr
iio_channel_get_all
iio_channel_get_all_cb
iio_channel_get_sys
ina209_update_device
ina2xx_update_device
inet_diag_lock_handler
inet_frag_find
inet_gso_segment
inflate
init_data_container
init_desc
init_inode_metadata
init_ipath
```



```
init_sdesc
inotify_new_group
input_devices_seq_start
input_handlers_seq_start
insert_with_overflow
instance_create
intel_framebuffer_create
intel_framebuffer_create_for_mode
intel_modeset_pipe_config
intel_user_framebuffer_create
internal_dev_create
ioctx_alloc
iommu_group_alloc
ip6addrlbl_alloc
ip6_dst_lookup_flow
ip6mr_vif_seq_start
ip6_sk_dst_lookup_flow
ip6t_register_table
ipath_alloc_devdata
ipath_alloc_fmr
ipath_alloc_pd
ipath_alloc_ucontext
ipath_create_ah
ipath_create_cq
ipath_create_qp
ipath_create_srq
ipath_get_dma_mr
ipath_reg_phys_mr
ipath_reg_user_mr
ipcctl_pre_down_nolock
ipc_lock
ipc_lock_check
ipc_obtain_object
ipc_obtain_object_check
ipddp_init
ip_make_skb
ipmr_mfc_seq_start
ipmr_rt_fib_lookup
ipmr_vif_seq_start
ipoib_add_port
ipoib_create_ah
__ip_route_output_key
ipt_register_table
__ip_tunnel_create
ipv6_add_addr
ipv6_gso_segment
ipv6_renew_options
iscsi_iser_ep_connect
iscsit_add_np
iscsit_add_tiqn
iscsit_tpg_add_network_portal
isert_device_find_by_ib_dev
isofs_export_get_parent
isofs_export_iget
isofs_iget
isofs_lookup
isp1760_register
iwch_ah_create
iwch_allocate_pd
iwch_alloc_fast_reg_mr
iwch_alloc_fastreg_pbl
iwch_alloc_mw
iwch_alloc_ucontext
iwch_create_cq
iwch_create_qp
iwch_register_phys_mem
```

```
iwch_reg_user_mr
iw_create_cm_id
iwl_drv_start
jbd2__journal_start
jc42_update_device
jffs2_acl_from_medium
jffs2_acl_to_medium
jffs2_add_physical_node_ref
jffs2_gc_fetch_inode
jffs2_get_acl
jffs2_iget
jffs2_lookup
jffs2_new_inode
jffs2_setup_xattr_datum
jffs2_write_dirent
jffs2_write_dnode
jfs_get_acl
jfs_iget
jfs_lookup
jfs_nfs_get_inode
journal_start
kern_path_create
kern_path_locked
key_alloc
key_create_or_update
key_get_instantiation_authkey
key_lookup
keyring_alloc
keyring_search
keyring_search_aux
__keyring_search_one
key_type_lookup
kmemleak_seq_start
kvm_create_vm
kvm_pfn_to_page
l2cap_create_basic_pdu
l2cap_create_connless_pdu
l2cap_create_iframe_pdu
l2cap_create_sframe_pdu
l2cap_sock_alloc_skb_cb
lbs_cfg_alloc
__lbs_cmd_async
__lbtf_cmd_async
lcd_device_register
lease_alloc
lio_target_call_addnptotpg
lio_target_call_coreaddtiqn
lio_target_make_nodeacl
lm75_update_device
lm80_update_device
load_msg
lockd_create_svc
lock_mount
logfs_get_dd_page
logfs_get_sb_device
__logfs_iget
logfs_mount
logfs_new_inode
logfs_new_meta_inode
logfs_read_meta_inode
lookup_bdev
lookup_dcache
__lookup_free_space_inode
lookup_one_len
lookup_real
lookup_user_key
```



```
lookup_vport
lookup_znode
loop_probe
ltc4151_update_device
ltc4261_update_device
ltree_add_entry
lzo_alloc_workspace
lzo_init
mac802154_add_iface
macvtap_get_socket
make_blackhole
make_cluster
make_comm
make_netconsole_target
make_node
make_receive_sock
_make_request
make_send_sock
make_space
map_extent_mft_record
map_mft_record_page
matrix_keypad_parse_dt
max1668_update_device
max6639_update_device
max6697_update_device
max77686_get_regmap
max77693_get_regmap
max8998_i2c_parse_dt_pdata
maybe_deliver
__mb_cache_entry_find
mca_cannibalize
md_import_device
mem_cgroup_css_alloc
memdup_user
mesh_path_add
mgag200_user_framebuffer_create
minix_iget
minix_lookup
__mkroute_output
mlx4_alloc_cmd_mailbox
mlx4_ib_alloc_fast_reg_mr
mlx4_ib_alloc_fast_reg_page_list
mlx4_ib_alloc_mw
mlx4_ib_alloc_pd
mlx4_ib_alloc_ucontext
mlx4_ib_alloc_xrcd
mlx4_ib_create_ah
mlx4_ib_create_cq
mlx4_ib_create_qp
mlx4_ib_create_srq
mlx4_ib_fmr_alloc
mlx4_ib_get_dma_mr
mlx4_ib_reg_user_mr
mlx5_alloc_cmd_msg
mlx5_ib_alloc_fast_reg_mr
mlx5_ib_alloc_fast_reg_page_list
mlx5_ib_alloc_pd
mlx5_ib_alloc_ucontext
mlx5_ib_alloc_xrcd
mlx5_ib_create_ah
mlx5_ib_create_cq
mlx5_ib_create_qp
mlx5_ib_create_srq
mlx5_ib_get_dma_mr
mlx5_ib_reg_user_mr
mm_access
```

```
mmc_alloc_card
mmc_blk_alloc_req
mmc_blk_ioctl_copy_from_user
mon_text_read_wait
mount_bdev
mount_fs
mount_mtd
mount_mtd_aux
mount_nodev
mount_ns
mount_pseudo
mount_single
mount_subtree
mount_subvol
mousedev_create
mpls_gso_segment
__mpol_dup
mpol_new
mqueue_get_inode
mqueue_mount
m_start
mthca_ah_create
mthca_alloc_fmr
mthca_alloc_mailbox
__mthca_alloc_mtt
mthca_alloc_pd
mthca_alloc_ucontext
mthca_create_cq
mthca_create_qp
mthca_create_srq
mthca_get_dma_mr
mthca_init_user_db_tab
mthca_reg_phys_mr
mthca_reg_user_mr
mwifiex_add_virtual_intf
nand_get_flash_type
nbd_find_request
ncp_lookup
__neigh_create
nes_alloc_fast_reg_mr
nes_alloc_fast_reg_page_list
nes_alloc_mw
nes_alloc_pd
nes_alloc_ucontext
nes_create_ah
nes_create_cq
nes_create_qp
nes_reg_phys_mr
nes_reg_user_mr
net2272_probe_init
netdev_create
netlink_getsockbyfilp
netlink_getsockbyportid
new_mountpoint
new_nbp
new_node_page
newpart
new_read
new_simple_dir
new_vmap_block
nfc_llcp_allocate_snl
__nf_conntrack_alloc
nfcsim_init_dev
nfs3_get_cached_acl
nfs3_proc_getacl
_nfs41_free_stateid
```



```
_nfs41_proc_sequence
nfs4_acl_posix_to_nfsv4
nfs4_alloc_client
nfs4_alloc_slot
nfs4_blk_decode_device
nfs4_blk_get_deviceinfo
nfs4blocklayout_register_sb
nfs4_create_referral_server
nfs4_create_sec_client
nfs4_create_server
nfs4_do_open
nfs4_do_unlck
nfs4_find_or_create_slot
nfs4_init_client
nfs4_insert_state_owner_locked
nfs4_label_alloc
_nfs4_opendata_reclaim_to_nfs4_state
_nfs4_opendata_to_nfs4_state
nfs4_open_recoverdata_alloc
nfs4_pathname_string
nfs4_proc_layoutget
nfs4_proc_lookup_mountpoint
nfs4_remote_referral_mount
nfs4_state_find_open_context
nfs4_try_open_cached
nfs_alloc_client
nfs_async_rename
nfs_callback_create_svc
nfs_clone_server
nfs_create_request
nfs_create_server
nfsd4_cld_register_sb
nfs_d_automount
nfsd_get_posix_acl
nfs_do_refmount
nfs_do_root_mount
nfs_do_submount
nfs_find_and_lock_request
nfs_follow_referral
nfs_follow_remote_path
nfs_found_client
nfs_fs_mount
nfs_get_lock_context
nfs_get_root
nfs_idmap_request_key
nfs_init_client
nfs_lookup
nf_sockopt_find
nfs_path
nfs_readdir_get_array
nfs_submount
nfs_try_mount_request
nfs_try_to_update_request
nilfs_get_dentry
nilfs_get_page
nilfs_get_parent
nilfs_iget
nilfs_iget_for_gc
nilfs_lookup
nilfs_mount
nilfs_new_inode
nl80211_parse_connkeys
__nlm_async_call
nlmclnt_init
nnode_lookup
notifier_err_inject_init
```

```
nouveau_connector_create
nouveau_gem_prime_import_sg_table
nouveau_gem_prime_vmap
nouveau_user_framebuffer_create
ntfs_attr_find_vcn_nolock
ntfs_attr_iget
ntfs_cluster_alloc
ntfs_get_parent
ntfs_iget
ntfs_index_iget
ntfs_lookup
ntfs_map_page
ntfs_mapping_pairs_decompress
ntfs_mft_record_alloc
ntfs_nfs_get_inode
ntfs_rl_realloc
ntfs_runlists_merge
nv04_pm_clocks_pre
nv40_pm_clocks_pre
nv50_pm_clocks_pre
nva3_pm_clocks_pre
nvc0_pm_clocks_pre
nvme_create_queue
nvme_map_user_pages
o2hb_heartbeat_group_make_item
o2hb_setup_one_bio
o2nm_cluster_group_make_group
o2nm_node_group_make_item
oaktrail_rfkill_new
objlayout_alloc_lseg
ocfs2_acl_from_xattr
ocfs2_acl_to_xattr
ocfs2_begin_quota_recovery
ocfs2_extend_local_quota_file
ocfs2_find_free_entry
ocfs2_get_acl
ocfs2_get_acl_nolock
ocfs2_get_dentry
ocfs2_get_parent
ocfs2_iget
ocfs2_iop_get_acl
ocfs2_local_quota_add_chunk
ocfs2_lookup
ocfs2_start_trans
ocfs2_zero_start_ordered_transaction
ocrdma_alloc_lkey
ocrdma_alloc_pd
ocrdma_alloc_ucontext
ocrdma_create_ah
ocrdma_create_cq
ocrdma_create_qp
ocrdma_create_srq
ocrdma_reg_user_mr
of_clk_get_by_name
of_get_fixed_voltage_config
of_get_gpio_regulator_config
of_node_to_pwmchip
of_pwm_get
of_pwm_simple_xlate
of_pwm_xlate_with_flags
__of_usb_find_phy
ohci_allocate_iso_context
omap_get_control_dev
omfs_find_entry
omfs_iget
omfs_lookup
```



```
omfs_new_inode
omfs_scan_list
open_exec
open_mtd_by_chdev
open_root_dentry
open_ubi
open_xa_dir
open_xa_root
opp_find_freq_ceil
opp_find_freq_floor
opp_get_notifier
osdmap_apply_incremental
osdmap_decode
osduld_info_lookup
osduld_path_lookup
ovs_dp_cmd_build_info
ovs_flow_actions_alloc
ovs_flow_alloc
ovs_flow_cmd_build_info
ovs_vport_add
ovs_vport_alloc
ovs_vport_cmd_build_info
oxu_create
p9_client_attach
p9_client_create
p9_client_getattr_dotl
p9_client_prepare_req
p9_client_rpc
p9_client_stat
p9_client_walk
p9_client_xattrwalk
p9_client_zc_rpc
p9_conn_create
p9_fid_create
p9_idpool_create
p9_tag_alloc
padata_get_next
parse_acl_data
parse_path
parse_priority_group
path_openat
pciserial_init_ports
pcpu_build_alloc_info
pcrypt_alloc
pcrypt_alloc_instance
pem_update_device
perf_cgroup_css_alloc
perf_event_alloc
perf_event_create_kernel_counter
perf_init_event
persistent_ram_new
pfkey_msg2xfrm_state
pfkey_xfrm_policy2msg_prep
__pfkey_xfrm_state2msg
phy_attach
phy_connect
platform_create_bundle
platform_device_register_full
pm_ctrl_init
pn533_send_cmd_sync
pnfs_layout_process
pnode_lookup
pool_create
__pool_find
posix_acl_from_disk
posix_acl_from_mode
```

```
posix_acl_from_xattr
posix_acl_to_disk
posix_state_to_acl
process_lvol
proc_lookup_de
proc_lookupfd_common
proc_map_files_lookup
proc_mount
proc_ns_dir_lookup
proc_ns_fget
proc_ns_follow_link
proc_ns_get_dentry
proc_pident_lookup
proc_pid_follow_link
proc_pid_lookup
proc_sys_lookup
proc_task_lookup
proc_tgid_net_lookup
psb_framebuffer_create
psb_user_framebuffer_create
pt1_alloc_adapter
ptm_unix98_lookup
ptp_clock_register
ptrace_get_task_struct
ptrace_register_breakpoint
pts_unix98_lookup
pwm_get
pwm_request
pwm_request_from_chip
qdisc_alloc
qdisc_get_stab
qib_alloc_devdata
qib_alloc_fast_reg_page_list
qib_alloc_pd
qib_alloc_ucontext
qib_create_ah
qib_create_cq
qib_create_qp
qib_create_qp0_ah
qib_create_srq
qib_init_iba6120_funcs
qib_init_iba7220_funcs
qib_init_iba7322_funcs
qib_reg_user_mr
qla1280_request_firmware
qla4xxx_ep_connect
qnx4_iget
qnx6_iget
qnx6_lookup
quotactl_block
r2hb_hb_group_make_item
r2nm_cluster_group_make_group
r2nm_node_group_make_item
radeon_gem_prime_import_sg_table
radeon_gem_prime_vmap
radeon_user_framebuffer_create
raid0_takeover
raid0_takeover_raid1
raid0_takeover_raid10
raid0_takeover_raid45
raid10_takeover
raid10_takeover_raid0
raid1_takeover
raid45_takeover_raid0
raid4_takeover
raid5_takeover
```



```
raid5_takeover_raid1
raid5_takeover_raid6
raid6_takeover
rbd_client_create
rbd_dev_v2_snap_name
rb_move_tail
rdma_alloc_frmr
rdma_create_id
__rds_conn_create
rds_ib_alloc_fmr
rds_ib_create_mr_pool
rds_ib_get_mr
rds_iw_alloc_mr
rds_iw_create_mr_pool
rds_iw_get_mr
rds_iw_map_scatterlist
rds_message_map_pages
read_add_inode
__read_cache_page
read_link
reg_copy_regd
reg_create
__register_chrdev_region
_register_divider
register_ip_vs_app
register_session
register_wide_hw_breakpoint
regmap_field_alloc
regmap_init
regmap_mmio_gen_context
_regulator_get
regulator_register
reg_umr
reiserfs_get_acl
reiserfs_get_page
reiserfs_get_parent
reiserfs_iget
reiserfs_lookup
remove_pmkid
request_asymmetric_key
request_key
request_key_and_link
request_key_auth_new
request_key_with_auxdata
reserve_sfa_size
reset_control_get
resolv_usage_page
rndis_alloc
rndis_alloc_inst
romfs_iget
romfs_lookup
romfs_mount
rom_init
__root_device_register
rpcauth_create
rpcauth_lookup_credcache
rpcb_create
rpc_bind_new_program
__rpc_clone_client
rpc_create
rpc_lookup
__rpc_lookup_create_exclusive
rpc_mkpipe_data
rpc_new_client
rpc_new_task
rpcrdma_create_id
```

```
rpc_verify_header
rp_find_vq
rqst_exp_find
rqst_exp_get_by_name
rtc_device_register
rtl8139_init_board
rtm_to_ifaddr
rtnl_create_link
rxrpc_accept_call
rxrpc_alloc_client_call
rxrpc_find_peer
rxrpc_get_bundle
rxrpc_get_client_call
rxrpc_get_null_key
rxrpc_get_peer
rxrpc_get_transport
rxrpc_incoming_call
rxrpc_incoming_connection
rxrpc_kernel_begin_call
rxrpc_lookup_local
rxrpc_name_to_transport
sbp_get_lun_from_tpg
sbp_make_nodeacl
sbp_make_tpg
sbp_make_tport
sbp_management_agent_register
sbp_session_create
sbp_target_agent_register
scan_for_dirty
scan_for_leb_for_idx
scan_get_nnode
scan_get_pnode
sch5627_update_device
sch5636_update_device
sched_create_group
scrub_setup_ctx
__scsi_add_device
scsi_devinfo_lookup_by_key
sctp_datamsg_from_user
sdhci_alloc_host
sdhci_pci_probe_slot
sdhci_pltfm_init
sdio_alloc_func
search_process_keyrings
securityfs_create_file
select_bad_process
select_one_root
sel_make_dir
sem_obtain_lock
seqiv_alloc
setup_conf
setup_essiv_cpu
setup_vq
sfi_check_table
sg_alloc
sget
sg_get_dev
shmem_file_setup
shmem_get_parent
shmem_read_mapping_page_gfp
simple_get_bytes
simple_get_netobj
simple_lookup
simple_transaction_get
__skb_gso_segment
skb_mac_gso_segment
```



```
skb_segment
skb_udp_tunnel_segment
skcipher_geniv_alloc
smb2_setup_async_request
smb2_setup_request
sm_get_zone
smm665_update_device
sock_alloc_file
solo_enc_alloc
split
split_token_from_name
squashfs_decompressor_init
squashfs_export_iget
squashfs_iget
squashfs_read_fragment_index_table
squashfs_read_id_index_table
squashfs_read_inode_lookup_table
squashfs_read_table
squashfs_read_xattr_id_table
squashfs_xz_init
srp_rport_add
srpt_make_nodeacl
srpt_make_tpg
srpt_make_tport
ssb_hcd_create_pdev
ssc_request
s_start
start_transaction
stmpe_keypad_of_probe
strndup_user
svc_bc_create_socket
svc_create_socket
svc_get_next_xprt
svc_prepare_thread
svc_rdma_create
svc_setup_socket
__svc_xpo_create
svm_create_vcpu
syscon_node_to_regmap
syscon_regmap_lookup_by_compatible
syscon_regmap_lookup_by_pdevname
syscon_regmap_lookup_by_phandle
sysctl_getname
sysctl_head_grab
sysfs_lookup
sysfs_mount
sysv_iget
sysv_lookup
sysv_new_inode
t4_uld_add
target_core_call_addhbatotarget
target_core_make_subdev
target_core_mappedlun_stat_mkdir
target_core_port_stat_mkdir
target_core_register_fabric
target_core_stat_mkdir
target_fabric_configfs_init
target_fabric_make_lun
target_fabric_make_mappedlun
target_fabric_make_nodeacl
target_fabric_make_np
target_fabric_make_tpg
target_fabric_make_wwn
tca6507_led_dt_init
tcf_action_get_1
tcf_action_init
```

```
1447 tcf_action_init_1
1448 tcf_hash_create
1449 tcm_loop_make_naa_tpg
1450 tcm_loop_make_scsi_hba
1451 tcm_qla2xxx_make_lport
1452 tcm_qla2xxx_make_nodeacl
1453 tcm_qla2xxx_make_tpg
1454 tcm_qla2xxx_npiv_make_lport
1455 tcm_qla2xxx_npiv_make_tpg
1456 tcp_tso_segment
1457 textsearch_prepare
1458 tgfx_probe
1459 thermal_cooling_device_register
1460 thermal_zone_device_register
1461 thermal_zone_get_zone_by_name
1462 timers_start
1463 tipc_alloc_conn
1464 tmp401_update_device
1465 tomoyo_get_absolute_path
1466 tomoyo_get_dentry_path
1467 tomoyo_get_local_path
1468 tracepoint_entry_add_probe
1469 tracepoint_entry_remove_probe
1470 tracepoint_remove_probe
1471 __tracing_open
1472 transport_init_session
1473 try_get_usb_function_instance
1474 try_location
1475 ttm_dma_pool_init
1476 __tty_alloc_driver
1477 tty_init_dev
1478 tty_ldisc_get
1479 tty_lookup_driver
1480 tty_open_current_tty
1481 tty_register_device_attr
1482 tun_alloc_skb
1483 tun_get_socket
1484 ubi_early_get_peb
1485 ubifs_get_nnode
1486 ubifs_get_pnode
1487 ubifs_iget
1488 ubifs_load_znode
1489 ubifs_lookup
1490 ubifs_lpt_lookup
1491 ubifs_lpt_lookup_dirty
1492 ubifs_mount
1493 ubifs_new_inode
1494 ubifs_read_sb_node
1495 ubifs_recover_leb
1496 ubifs_recover_log_leb
1497 ubifs_scan
1498 ubifs_start_scan
1499 ubifs_tnc_next_ent
1500 ubi_open_volume
1501 ubi_open_volume_nm
1502 ubi_open_volume_path
1503 _ucma_find_context
1504 udf_get_parent
1505 udf_lookup
1506 udf_nfs_get_inode
1507 udl_fb_user_fb_create
1508 udl_gem_prime_import
1509 udp4_ufo_fragment
1510 udp6_ufo_fragment
1511 ufs_get_locked_page
1512 ufs_get_page
```



```
1513 ufs_get_parent
1514 ufs_iget
1515 ufs_lookup
1516 ufs_new_inode
1517 ufs_nfs_get_inode
1518 u_memcpya
1519 unix_gid_find
1520 __unlink_start_trans
1521 unpack_dfa
1522 unpack_profile
1523 unx_create_cred
1524 update_pmkid
1525 usb_cdc_wdm_register
1526 __usb_find_phy
1527 __usb_find_phy_dev
1528 usb_get_function_instance
1529 usb_gstrings_attach
1530 use_block_rsv
1531 user_path_parent
1532 uvesafb_vbe_state_save
1533 uwb_rc_neh_add
1534 V1_minix_iget
1535 V2_minix_iget
1536 v4l2_clk_find
1537 v4l2_clk_register
1538 v4l2_m2m_ctx_init
1539 v4l2_m2m_init
1540 v9fs_create
1541 v9fs_fid_lookup_with_uid
1542 __v9fs_get_acl
1543 v9fs_get_inode
1544 v9fs_mount
1545 v9fs_qid_iget
1546 v9fs_qid_iget_dotl
1547 v9fs_session_init
1548 v9fs_vfs_lookup
1549 v9fs_writeback_fid
1550 vb2_dc_alloc
1551 vb2_dc_attach_dmabuf
1552 vb2_dc_dmabuf_ops_map
1553 vb2_dc_get_userptr
1554 vb2_dma_contig_init_ctx
1555 vb2_vmalloc_attach_dmabuf
1556 verdict_instance_lookup
1557 vfat_lookup
1558 vfio_create_group
1559 vfio_group_create_device
1560 vfio_iommu_type1_open
1561 vfs_kern_mount
1562 vhost_net_ubuf_alloc
1563 video_output_register
1564 virtio_find_single_vq
1565 vma_to_resize
1566 vmci_ctx_create
1567 vm_setup_vq
1568 vmstat_start
1569 vmw_kms_fb_create
1570 vmx_create_vcpu
1571 vxfs_get_page
1572 __vxfs_iget
1573 vxfs_iget
1574 vxfs_lookup
1575 vxlan_socket_create
1576 wait_on_page_read
1577 wakelock_lookup_add
1578 wil_cfg80211_init
```

```
wil_if_alloc
wimax_gnl_re_state_change_alloc
wimax_msg_alloc
wl1251_alloc_hw
wl12xx_get_platform_data
wlcore_alloc_hw
x509_cert_parse
xattr_lookup
xen_blkif_alloc
xenbus_dev_request_and_reply
xen_copy_pss_data
xennet_create_dev
xenvif_alloc
xfrm6_dst_lookup
xfrm_alloc_dst
xfrm_bundle_lookup
__xfrm_dst_lookup
xfrm_lookup
xfrm_policy_lookup_bytype
xfrm_policy_netlink
xfrm_resolve_and_create_bundle
xfrm_sk_policy_lookup
xfrm_state_netlink
xfs_acl_from_disk
xfs_fs_get_parent
xfs_get_acl
xfs_handle_to_dentry
xfs_nfs_get_inode
xfs_vn_ci_lookup
xfs_vn_lookup
xlog_alloc_log
xpc_create_gru_mq_uv
xprt_create_transport
xprt_dynamic_alloc_slot
xprt_setup_rdma
xs_create_sock
xs_setup_bc_tcp
xs_setup_local
xs_setup_tcp
xs_setup_udp
xs_setup_xprt
xs_talkv
xt_find_match
xt_find_table_lock
xt_find_target
xt_hook_link
xt_register_table
zlib_alloc_workspace
zlib_init
```

```
**********************************************************
      45 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.returns_err_ptr.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 jedec_probe
   2 find_inode
   3 lookup_extent_mapping
```

```
**********************************************************
     430 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.returns_held_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of functions that return a held device.
// generated by 'gen_returns_held.sh'
dev_get_by_index
dev_get_by_macvtap_minor
dev_get_by_name
dev_to_net_device
dn_dev_get_default
find_lec_by_itfnum
get_vlan
ieee802154_get_dev
ieee802154_nl_get_dev
ip6mr_reg_vif
__ip_dev_find
ipmr_new_tunnel
ipmr_reg_vif
mac802154_add_iface
nr_ax25_dev_get
nr_dev_first
nr_dev_get
phonet_device_get
phonet_route_output
rose_dev_get
x25_dev_get
```

```
**********************************************************
    1575 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.rosenberg_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of copy_to_user function and buffer parameters.
// generated by 'gen_rosenberg_funcs.sh'
bits_to_user 0
cfs_trace_copyout_string 2
compat_copy_entry_to_user 0
compat_filldir 1
compat_filldir64 1
copy_gctl_to_user 2
copy_msqid_to_user 1
copy_semid_to_user 1
copy_shmid_to_user 1
copy_shminfo_to_user 1
copy_to 1
__copy_to_user 1
copy_to_user 1
crystalhd_user_data 1
csum_partial_copy_to_user 1
diva_os_copy_to_user 1
diva_os_copy_to_user 2
divas_maint_read 1
divas_maint_write 1
divas_um_idi_copy_to_user 1
drm_copy_field 2
echo_copyout_lsm 0
fat_compat_ioctl_filldir 1
fat_ioctl_filldir 1
fd_copyout 1
filldir 1
filldir64 1
gnet_stats_copy 2
gnet_stats_copy_app 1
gnet_stats_copy_queue 1
gru_user_copy_handle 1
ib_copy_to_udata 1
kernel_termios_to_user_termios 1
kernel_termios_to_user_termios_1 1
kfifo_copy_to_user 1
__kfifo_to_user 1
libcfs_ioctl_popdata 1
maint_read_write 0
memcpy_toiovec 1
memcpy_toiovecend 1
mic_virtio_copy_to_user 1
mic_vringh_copy 2
nla_put 3
nla_put_string 2
nvme_trans_copy_to_user 1
nvme_trans_supported_vpd_pages 2
nvme_trans_unit_serial_page 2
obd_ioctl_popdata 1
putused_user 1
put_v4l2_input32 0
put_v4l2_pix_format 0
put_v4l2_pix_format_mplane 0
put_v4l2_sliced_vbi_format 0
put_v4l2_vbi_format 0
seq_copy_in_user 1
set_arg 1
set_fd_set 2
str_to_user 0
```



```
uinput_str_to_user 1
__videobuf_copy_stream 1
__videobuf_copy_to_user 1
xdi_copy_to_user 1
xdi_copy_to_user 2
xfer_to_user 1
xfrm_mark_put 1
xfs_bulkstat_one_fmt 3
xfs_getbmap_format 1
xfs_getbmapx_format 1
xfs_inumbers_fmt 1
xt_compat_match_to_user 0
xt_compat_target_to_user 0
```

**********
*267 Fri Dec 21 15:00:23 2018*
*new/usr/src/tools/smatch/src/smatch_data/kernel.silenced_functions*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/* Don't print anything from these functions */
atomic_dec_and_test
atomic_inc_and_test
atomic64_dec_and_test
atomic_sub_and_test
test_and_clear_bit
test_and_set_bit
__copy_to_user_nocheck
__copy_from_user_nocheck
arch_static_branch
__static_cpu_has
__read_once_size
```

```
**********************************************************
    9892 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.sizeof_param
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of function parameters that are the size of a buffer.
// generated by 'gen_sizeof_param.sh'
access_process_vm 3 2
acpi_os_unmap_memory 1 0
add_device_randomness 1 0
add_numbered_child 4 3
adpt_i2o_post_wait 2 1
adpt_i2o_query_scalar 5 4
adv7170_write_block 2 1
adv7175_write_block 2 1
af9013_write_ofsm_regs 3 2
af9015_read_regs 3 2
af9033_rd_regs 3 2
anysee_ctrl_msg 2 1
anysee_ctrl_msg 4 3
asd_read_flash_seg 3 1
asd_read_ocm_seg 3 1
async_set_registers 2 3
at76_get_mib 3 2
ata_dev_config_ncq 2 1
ata_exec_internal 5 4
ata_id_c_string 3 1
ath10k_dbg_dump 4 3
ath10k_pci_diag_write_mem 3 2
ath6kl_init_get_fwcaps 2 1
ath6kl_wmi_add_wow_pattern_cmd 3 5
ath9k_hw_name 2 1
atmel_copy_to_card 3 2
bch_bkey_to_text 1 0
bch_btree_to_text 1 0
be_roce_mcc_cmd 2 1
be_wrb_cmd_hdr_prepare 3 0
be_wrb_hdr_prepare 1 0
bitmap_scnprintf 1 0
blkg_path 2 1
bluecard_read 3 2
bnep_send 2 1
bnx2fc_initiate_els 3 2
brcmf_fil_bsscfg_data_set 3 2
brcmf_fil_iovar_data_get 3 2
bt819_write_block 2 1
bt_get_result 2 1
btrfs_insert_item 4 3
bt_start_transaction 2 1
__builtin_memset 2 0
bulk_w 3 2
calc_hmac 2 1
calc_mic_tx_frag 2 1
ceph_osdmap_state_str 1 0
ceph_x_encrypt 4 3
cfg80211_get_p2p_attr 4 3
cfg80211_tx_mlme_mgmt 2 1
ckhdid_printf 1 0
__clear_user 1 0
clear_user 1 0
cmd_complete 5 4
cmsghdr_from_user_compat_to_kern 3 2
_config_request 5 4
console_cont_flush 1 0
__copy_from_user 2 0
```



```
__copy_from_user 2 0
__copy_from_user_inatomic 2 0
copy_from_user_nmi 2 0
copy_in_user 2 0
copy_in_user 2 1
__copy_to_user 2 1
copy_to_user 2 1
cpulist_scnprintf 1 0
cpumask_scnprintf 1 0
create_syslog_header 2 1
csio_enqueue_evt 3 2
csio_hostname 1 0
csio_osname 1 0
csum_partial 1 0
cx18_eeprom_dump 2 1
cxd2820r_rd_regs 3 2
cxusb_ctrl_msg 3 2
cyapa_i2c_reg_write_block 2 3
cyttsp4_adap_read 2 3
dbg_chgconf 2 1
dbg_chgstat 1 0
dbg_command_buf 1 0
dbg_intr_buf 1 0
dbg_port_buf 1 0
dbg_regstat 1 0
dbg_status_buf 1 0
derived_key_decrypt 2 1
derived_key_encrypt 2 1
dgrp_dpa 2 1
dgrp_monitor 2 1
__d_head 1 0
digsig_verify_rsa 4 3
DIVA_DIDD_Read 1 0
dma_free_attrs 1 2
dma_map_single_attrs 2 1
dmi_format_ids 1 0
domain_flush_cache 2 1
d_path 2 1
ds2780_battery_io 3 1
ds2780_write 3 1
ds2781_battery_io 3 1
ds2781_write 3 1
ds_recv_data 2 1
ds_recv_status_nodump 3 2
dst_put_ci 2 1
dump_emit 2 1
dvb_play_kernel 2 1
early_iounmap 1 0
ec_transaction 2 1
edac_dimm_info_location 2 1
edt_ft5x06_ts_readwrite 1 2
edt_ft5x06_ts_readwrite 3 4
efx_mcdi_rpc 3 2
efx_mcdi_rpc 5 4
efx_mcdi_rpc_finish 4 3
enc28j60_mem_read 2 3
evergreen_hdmi_update_avi_infoframe 2 1
fast_mix 2 1
fcoe_wwn_to_str 2 1
fill_note 3 4
frag_safe_skb_hp 2 3
fw_csr_string 3 2
garmin_write_bulk 2 1
get_atrandom_bytes 1 0
get_fw_name 2 1
get_jack_mode_name 3 2
```

```
get_modalias 2 1
get_name 3 1
get_random_bytes 1 0
get_registers 2 3
get_rhf_errstring 2 1
gnet_stats_copy 3 2
gru_get_cb_exception_detail_str 3 2
gru_send_message_gpa 2 1
gspca_frame_add 3 2
h5_link_control 2 1
__hci_cmd_sync 2 3
hci_req_add 2 3
hdmi_audio_infoframe_pack 2 1
hdmi_avi_infoframe_pack 2 1
hdmi_print_pcm_rates 2 1
hex_dump_to_buffer 1 0
hex_dump_to_buffer 5 4
hfa384x_from_aux 2 3
hfa384x_from_bap 3 2
hfa384x_to_aux 2 3
hp_wmi_perform_query 3 2
__hw_addr_add 2 1
i2400m_bm_cmd 2 1
i2400m_msg_check_status 2 1
i2400m_msg_to_dev 2 1
__i2400mu_send_barker 2 1
i2c_master_recv 2 1
i2c_master_send 2 1
i2c_read_demod_bytes 3 2
i2c_read_eeprom 4 3
i2c_smbus_read_i2c_block_data 2 3
i2c_smbus_write_i2c_block_data 2 3
i2c_w 3 2
i2c_write_demod_bytes 2 1
i2o_parm_issue 3 2
i2o_parm_table_get 7 6
ib_copy_to_udata 2 1
ic_bootp_string 3 0
init_cdrom_command 2 1
input_bits_to_string 1 0
intel_sdvo_get_value 3 2
intel_sdvo_write_infoframe 4 3
__iommu_flush_cache 2 1
ipath_decode_err 2 1
__ipr_format_res_path 2 1
ipr_format_res_path 3 2
ip_vs_dbg_addr 2 1
ip_vs_dbg_callid 1 0
ipw2100_get_fwversion 2 1
ipw2100_get_ucodeversion 2 1
ipw_send_cmd_pdu 2 3
irnet_read_discovery_log 2 1
iscsi_if_send_reply 6 5
isdn_tty_getdial 2 1
isl12022_read_regs 3 2
it913x_io 7 6
it913x_read_reg 3 2
iwl_dvm_send_cmd_pdu 3 4
iwl_mvm_send_cmd_pdu 3 4
jdvbt90502_reg_read 3 2
kdb_getstr 1 0
key_get_type_from_user 2 0
kmemcheck_mark_initialized 1 0
kmemdup 1 0
kmsg_dump_get_line_nolock 3 2
ks8995_read 3 1
```



```
kvm_read_guest_cached 3 2
kvm_read_guest_page_mmu 5 3
kvm_read_nested_guest_page 4 2
l2cap_send_cmd 3 4
ldm_get_vstr 2 1
line6_read_data 3 2
lme2510_usb_talk 2 1
lme2510_usb_talk 4 1
lme2510_usb_talk 4 3
logfs_crc32 1 0
logfs_inode_write 2 1
lpfc_nlp_state_name 1 0
match_strlcpy 2 0
mce_async_out 2 1
memchr 2 0
memcmp 2 0
memcmp 2 1
memcpy 2 0
memcpy 2 1
memcpy_fromio 2 0
memcpy_toio 2 1
memdup_user 1 0
memmove 2 0
memmove 2 1
memset 2 0
memset_io 2 0
mgmt_event 3 2
mgmt_exec_nonemb_cmd 3 2
mgmt_pending_add 4 3
mlx5_cmd_exec 2 1
mlx5_cmd_exec 4 3
mlx5_core_create_mkey 3 2
mlx5_core_eq_query 3 2
mlx5_core_qp_modify 4 3
mlx5_core_qp_query 3 2
mma8450_read_block 3 2
modecpy 2 1
mpol_to_str 1 0
msg_print_text 4 3
mt2060_writeregs 2 1
mt2131_writeregs 2 1
mt2266_writeregs 2 1
mt312_read 3 2
mt312_write 3 2
_mt352_write 2 1
mt352_write 2 1
musb_write_fifo 1 2
mwifiex_copy_rates 3 2
mwl8k_cmd_name 2 1
mwl8k_send_fw_load_cmd 2 1
__mxt_read_reg 2 3
my_hd 1 0
ncp_add_mem 2 1
next_entry 2 0
nf_nat_ftp_fmt_cmd 3 2
nla_memcmp 2 1
nla_put 2 3
nla_strlcpy 2 0
nlmsg_perm 3 2
nouveau_pm_perflvl_info 2 1
o2hb_debug_create 5 7
o2hb_fill_node_map 1 0
o2hb_fill_node_map_from_callback 1 0
o2net_fill_node_map 1 0
o2net_sendpage 2 1
o2nm_configured_node_map 1 0
```

```
ocfs2_sprintf_system_inode_name 1 0
ocrdma_copy_cpu_to_le32 2 0
ocrdma_le32_to_cpu 1 0
or51132_writebuf 2 1
orinoco_get_wpa_ie 1 0
oz_get_next_device_name 2 1
p54spi_spi_read 3 2
PC4500_readrid 3 2
PC4500_writerid 3 2
pcan_usb_pro_send_req 4 3
pci_free_consistent 1 2
pci_map_single 2 1
pci_read_vpd 2 3
pdu_read 2 1
pep_reply 4 3
persistent_ram_decode_rs8 2 1
picolcd_send_and_wait 3 2
pidff_find_fields 3 1
pidff_find_special_keys 3 2
platform_device_add_data 2 1
pn_raw_send 1 0
print_hex_dump 6 5
print_hex_dump_bytes 3 2
printvalue 2 1
proc_get_long 5 4
psb_intel_sdvo_get_value 3 2
put_frag 2 1
pvr2_hdw_report_clients 2 1
pvr2_hdw_report_unlocked 3 2
pvr2_ioread_set_sync_key 2 1
pvr2_std_id_to_str 1 0
qlt_sched_sess_work 3 2
ql_write_cfg 2 1
queue_event 3 2
qword_get 2 1
r2hb_fill_node_map_from_callback 1 0
r2net_sendpage 2 1
r600_hdmi_update_audio_infoframe 2 1
r600_hdmi_update_avi_infoframe 2 1
r820t_read 3 2
r820t_write 3 2
rdcat 3 2
read_rom 2 3
_recv 2 1
reg_w 3 2
reg_wb 4 3
reg_w_buf 2 1
reg_w_ixbuf 3 2
reg_w_var 3 2
rndis_set_oid 3 2
root_nfs_cat 2 0
rpc_ntop 2 1
rpc_ntop4 2 1
rpc_ntop6_noscopeid 2 1
rsxx_creg_read 2 3
rtl8723ae_fill_h2c_cmd 2 3
rtl88e_fill_h2c_cmd 2 3
rtl92c_fill_h2c_cmd 2 3
rtl92d_fill_h2c_cmd 2 3
rtn_scope 1 0
rtn_type 1 0
s35390a_get_reg 3 2
s35390a_set_reg 3 2
saa7110_write_block 2 1
saa712x_write_regs 3 1
saa7185_write_block 2 1
```



```
scnprintf 1 0
scnprint_id 2 1
scsi_execute_req 4 3
scsi_mode_sense 4 3
scsi_sg_copy_from_buffer 2 1
sctp_addto_chunk 1 2
sctp_sf_abort_violation 6 5
_send 2 1
send_bulk_static_data 2 1
sendcmd_withirq 3 2
seq_write 2 1
set_registers 2 3
sg_copy_from_buffer 3 2
sg_copy_to_buffer 3 2
sg_init_one 2 1
sg_set_buf 2 1
si476x_cmd_tune_seek_freq 3 2
si476x_cmd_tune_seek_freq 5 4
si476x_core_i2c_xfer 3 2
simple_read_from_buffer 4 3
sirdev_raw_write 2 1
skb_copy_to_linear_data 2 1
skb_header_pointer 2 3
sky2_name 2 1
smp_send_cmd 2 3
snd_hda_get_pin_label 4 3
snd_info_get_line 2 1
snd_info_get_str 2 0
snd_midi_event_decode 2 1
snd_mixart_send_msg 2 3
snd_pcm_debug_name 2 1
snd_print_channel_allocation 2 1
snd_print_pcm_bits 2 1
snd_rawmidi_kernel_read 2 1
snd_rawmidi_transmit_peek 2 1
snd_seq_expand_var_event 1 2
snd_task_name 2 1
snd_usb_caiaq_send_command 3 2
snd_usb_ctl_msg 7 6
snprint_line 1 0
snprint_time 1 0
sock_kfree_s 2 1
solo_p2m_dma 4 2
sort 2 0
spi_read 2 1
spi_write 2 1
spi_write_then_read 2 1
spi_write_then_read 4 3
sprint_oid 3 2
squashfs_read_metadata 4 1
string_get_size 3 2
stringify_lockname 3 2
strlcat 2 0
strlcpy 2 0
strncat 2 0
strncmp 2 0
strncmp 2 1
strncpy 2 0
strncpy_from_user 2 0
__svc_print_addr 2 1
svc_print_addr 2 1
svc_print_xprts 1 0
sw_3dp_id 2 1
swap_dws 1 0
__sym_mfree 2 1
synaptics_rmi4_i2c_block_read 3 2
```

```
391 t4_wr_mbox 3 2
392 tcp_fastopen_reset_cipher 1 0
393 tda1004x_write_buf 3 2
394 tda18212_wr_regs 3 2
395 tda8083_readregs 3 2
396 textify_hooks 1 0
397 tipc_media_addr_printf 1 0
398 tomoyo_addprintf 1 0
399 tomoyo_print_ip 1 0
400 tomoyo_print_ipv4 1 0
401 tomoyo_print_ipv6 1 0
402 tomoyo_print_ulong 1 0
403 to_shortname_char 2 1
404 tpm_inf_send 2 1
405 tps6586x_reads 2 3
406 tps6586x_writes 2 3
407 trace_brcms_txdesc 2 1
408 trace_i915_reg_rw 3 2
409 trace_note 4 3
410 ttsp_write_block_data 2 3
411 ttusb2_msg 5 4
412 ttusb_cmd 2 1
413 ttusb_dec_send_command 2 3
414 ttusb_result 2 1
415 tuner_i2c_xfer_send 2 1
416 tveeprom_read 2 1
417 tw_transfer_internal 3 2
418 ubi_io_read 4 1
419 unicode_to_ascii 1 0
420 usb_bulk_msg 3 2
421 usb_control_msg 7 6
422 usb_fill_bulk_urb 4 3
423 usb_get_descriptor 4 3
424 usb_make_path 2 1
425 usbnet_read_cmd 6 5
426 usb_stor_set_xfer_buf 1 0
427 usb_string 3 2
428 uwb_dev_addr_print 1 0
429 uwb_mac_addr_print 1 0
430 uwb_rc_cmd 3 2
431 uwb_rc_vcmd 3 2
432 valid_stack_ptr 2 1
433 vic_provinfo_add_tlv 2 3
434 vpx3220_write_block 2 1
435 vscnprintf 1 0
436 w1_ds2760_write 3 1
437 w1_write_block 2 1
438 wbuf_read 2 3
439 wiimote_cmd_write 3 2
440 wiimote_queue 2 1
441 wil_memcpy_fromio_32 2 0
442 wimax_addr_scnprint 1 0
443 wl1251_cmd_configure 3 2
444 wl1251_cmd_interrogate 3 2
445 wl1251_cmd_send 3 2
446 wl1251_mem_read 3 2
447 wl1251_read_eeprom 3 2
448 wl1271_cmd_configure 3 2
449 wl1271_cmd_interrogate 3 2
450 wl1271_cmd_send 3 2
451 wl1271_cmd_template_set 4 3
452 wl1271_cmd_test 2 1
453 wl3501_set_to_wla 3 2
454 wlcore_cmd_configure_failsafe 3 2
455 wlcore_read 3 2
456 wlcore_write 3 2
```



```
457 write_extent_buffer 3 1
458 write_rom 2 3
459 wsm_read_mib 3 2
460 wsm_write_mib 3 2
461 wusb_key_dump 1 0
462 wusb_prf_256 1 0
463 wusb_prf_256 6 5
464 xc_send_i2c_data 2 1
465 xfrm_dst_alloc_copy 2 1
466 xor8_buf 1 0
467 yealink_set_ringtone 2 1
468 zl10036_write 2 1
469 zl10039_write 3 2
470 zl10353_write 2 1
471 zr36050_pushit 2 3
472 zr36060_pushit 2 3
```

```
**********************************************************
     181 Fri Dec 21 15:00:23 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.sizeof_param.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
printk
find_first_zero_bit
dvb_usb_generic_rw
__dynamic_pr_debug
strncmp
strncpy 2 1
strlcpy 2 1
strncat 2 1
snprintf
copy_from_user 2 1
copy_to_user 2 0
read_rom 2 3
skb_pull 1 -1
```

```
**********************************************************
     157 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.unconstant_macros
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 X86_VM_MASK
   2 VM_GROWSUP
   3 VM_SOFTDIRTY
   4 UBIFS_BLOCKS_PER_PAGE
   5 DYNTICK_TASK_NEST_MASK
   6 ARCH_SHF_SMALL
   7 SLAB_NOTRACK
   8 FAULT_FLAG_TRANSHUGE
   9 _PAGE_NX
  10 MAP_UNINITIALIZED
```

```
**********************************************************
     566 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.unreachable.ignore
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 list_for_each
   2 list_for_each_entry
   3 list_for_each_entry_safe
   4 list_for_each_entry_rcu
   5 list_for_each_entry_continue
   6 list_for_each_entry_continue_rcu
   7 hlist_for_each_entry
   8 hlist_for_each_entry_safe
   9 ax25_uid_for_each
  10 inet_twsk_for_each_inmate
  11 udp_portaddr_for_each_entry_rcu
  12 hlist_for_each_safe
  13 hlist_for_each_entry_rcu
  14 snd_pcm_group_for_each_entry
  15 DLM_ASSERT
  16 ata_for_each_dev
  17 netlbl_af4list_foreach_rcu
  18 netlbl_af6list_foreach_rcu
  19 for_primary_ifa
  20 sk_nulls_for_each
  21 unreachable
  22 iterate_all_kinds
  23 netdev_for_each_lower_dev
  24 for_each_clear_bit_from
  25 idr_for_each_entry_continue
```

```
**********************************************************
      22 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.unreachable.turn_off
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 BUG
2 panic
3 LBUG
4 ASSERT
```

```
**********************************************************
   10205 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/kernel.unwind_functions
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of unwind functions.
// generated by 'gen_unwind_functions.sh'
_aac_reset_adapter
aac_sa_init
__aac_shutdown
ace_init_cleanup
acpi_os_remove_interrupt_handler
ad5421_probe
ad5421_remove
ad5504_probe
ad5504_remove
ad714x_remove
ad7150_probe
ad7150_remove
ad7280_remove
ad7291_probe
ad7291_remove
ad7606_probe
ad7606_remove
ad7816_probe
ad7816_remove
ad7877_probe
ad7879_remove
ad799x_probe
ad799x_remove
adapter_down
adapter_init
adapter_uninit
adis_probe_trigger
adis_remove_trigger
adm8211_probe
adm8211_remove
adp5588_irq_teardown
adp5588_probe
adp5588_remove
adp5589_probe
adp5589_remove
adpt_i2o_delete_hba
ads7846_probe
adt7x10_remove
adv7180_remove
adxl34x_remove
aer_remove
ahc_platform_free
ahd_platform_free
airo_close
ali_ircc_close
altera_jtaguart_shutdown
altera_uart_shutdown
amd8111e_close
amd8111e_open
apds990x_remove
arcmsr_remove
arcrimi_found
arizona_free_irq
arizona_irq_exit
ark_pci_probe
ark_pci_remove
asd_pci_probe
asd_pci_remove
```



```
ast_driver_unload
at86rf230_probe
at86rf230_remove
ath10k_pci_probe
ath10k_pci_remove
ath5k_deinit_ah
ath5k_init_ah
ath5k_pci_probe
ath5k_pci_remove
atl1c_free_irq
atl1_down
atl1e_free_irq
atl1_probe
atl1_remove
atl2_free_irq
atp870u_probe
aty_disable_irq
axnet_close
azx_free
b43legacy_wireless_core_stop
b43_wireless_core_stop
b44_close
bcma_host_pci_probe
bcma_host_pci_remove
bdx_hw_stop
beiscsi_quiesce
be_unmap_pci_bars
bfad_pci_uninit
bfad_remove_intr
bh1770_remove
blogic_relres
bma150_remove
bnx2i_free_hba
bnx2_init_board
bnx2_init_one
bnx2x_free_irq
__bnx2x_remove
brcmf_sdio_intr_unregister
brcms_free
bu21013_free_irq
c2_probe
c2_register_device
c2_remove
c2_unregister_device
c4_add_dev
ca91cx42_irq_exit
cafe_nand_probe
cafe_nand_remove
cafe_pci_probe
cafe_shutdown
carm_init_one
carm_remove_one
cas_init_one
cas_remove_one
cb710_suspend
cc770_close
c_can_close
c_can_pci_probe
c_can_pci_remove
chd_dec_disable_int
cleanup_dev
cma3000_exit
cma3000_init
cmos_do_probe
com20020_found
com90io_found
```

```
cp_tm1217_probe
cp_tm1217_remove
cros_ec_register
cros_ec_remove
csio_intr_disable
cw1200_sdio_irq_unsubscribe
cw1200_spi_irq_subscribe
cw1200_spi_irq_unsubscribe
cx23885_finidev
cx25821_finidev
cx8800_finidev
cx8800_initdev
cx8802_fini_common
cxgb_down
cy8ctmg110_probe
cy8ctmg110_remove
cyttsp4_remove
cyttsp_remove
da9052_free_irq
DAC960_DetectCleanup
dc390_probe_one
de4x5_close
deassign_host_irq
deinit_card
denali_irq_cleanup
device_irq_exit
devm_free_irq
__devm_release_region
dfx_close
dfx_open
disable_igfx_irq
diva_os_remove_irq
dm1105_probe
dm1105_remove
dmfe_init_one
ds1307_remove
dt3155_probe
dt3155_remove
dw_spi_add_host
dw_spi_remove_host
e1000_free_irq
e1000_test_msi_interrupt
e100_down
e100_probe
e100_remove
eata2x_release
edt_ft5x06_ts_probe
edt_ft5x06_ts_remove
efx_nic_fini_interrupt
ems_pcmcia_del_card
enc28j60_probe
enc28j60_remove
enic_free_intr
epic_init_one
epic_remove_one
ess_dsp_init
ethoc_stop
fcpcipnp_release
fdomain_16x0_release
fealnx_init_one
fealnx_remove_one
flexcop_pci_exit
flexcop_pci_init
fnic_free_intr
fore200e_shutdown
fpga_probe
```



```
fpga_remove
free_dmar_iommu
free_irq_resources
free_region
fsa9480_probe
fsa9480_remove
gether_cleanup
goku_remove
gp2a_probe
gp2a_remove
gpio_remove_key
hdlcdrv_unregister
he_stop
hostap_remove_interface
hp100_close
hpc_release_ctlr
hpet_ioctl_common
hpwdt_exit
hpwdt_init_one
hw_card_shutdown
i2400m_release
i2c_hid_probe
i2c_hid_remove
i2o_pci_irq_disable
i740fb_probe
i740fb_remove
i801_probe
i801_remove
i82092aa_pci_probe
i82092aa_pci_remove
i915_driver_load
i915_driver_unload
ide_unregister
if_cs_release
if_spi_probe
igb_free_irq
iio_trigger_detach_poll_func
il3945_pci_remove
ili210x_i2c_probe
ilo_map_device
ilo_probe
ilo_remove
ilo_unmap_device
init_card
initio_probe_one
initio_remove_one
init_irq
inv_mpu6050_remove_trigger
ioh_gpio_probe
ioh_gpio_remove
ipath_ht_free_irq
ipg_probe
ipg_remove
ipr_test_msi
ipw2100_pci_init_one
ipw2100_pci_remove_one
ipw_pci_probe
ipw_pci_remove
ipw_prom_free
isdn_net_realrm
isl1208_probe
isl1208_remove
ixgb_down
ixgbe_free_irq
ixgbe_sfp_detection_subtask
jme_free_irq
```

```
ks8851_remove
ks_net_stop
kvaser_pci_init_one
kxtj9_probe
kxtj9_remove
libertas_spi_remove
lis3lv02d_joystick_disable
lm8323_remove
lm8333_probe
lm8333_remove
lmc_ifdown
lola_free
lp8727_release_irq
lp8788_irq_exit
lpfc_sli4_disable_intr
lpfc_sli4_disable_msi
lpfc_sli_disable_intr
lpfc_sli_disable_msi
marvell_pata_active
matroxfb_disable_irq
max11801_ts_probe
max11801_ts_remove
max17042_remove
max732x_irq_teardown
max7359_probe
max7359_remove
max77686_irq_exit
max77693_irq_exit
max8925_device_exit
max8997_irq_exit
max8998_irq_exit
mc13xxx_common_cleanup
mcs5000_ts_probe
mcs5000_ts_remove
mcs_touchkey_probe
mcs_touchkey_remove
megaraid_fini_mbox
megaraid_init_mbox
__megaraid_shutdown
mei_me_probe
mei_me_remove
mga_vram_init
mgsl_release_resources
microread_i2c_probe
microread_i2c_remove
middma_shutdown
mid_setup_dma
mlx4_en_destroy_netdev
mlx4_free_irqs
mm_pci_remove
mpr_touchkey_probe
mpr_touchkey_remove
mpt_adapter_dispose
mpu3050_probe
mpu3050_remove
mrf24j40_remove
mthca_free_irqs
musb_free
mvumi_unmap_pci_addr
mwl8k_probe
mwl8k_remove
mxser_board_remove
mxt_probe
ne2k_pci_close
nes_probe
nes_remove
```



```
net2272_probe_fin
net2272_remove
net2280_remove
net_close
netdev_close
netdev_free
ngene_start
ngene_stop
nj_release
notifier_del_irq
nozomi_card_exit
ns83820_init_one
nsc_ircc_close
ns_init_card_error
oaktrail_hdmi_i2c_exit
ohci_stop
orinoco_nortel_init_one
orinoco_nortel_remove_one
orinoco_pci_init_one
orinoco_pci_remove_one
orinoco_pci_suspend
orinoco_plx_init_one
orinoco_plx_remove_one
orinoco_tmd_init_one
orinoco_tmd_remove_one
parport_pc_unregister_port
pcan_free
pcf50633_irq_free
pcf8574_kp_probe
pcf8574_kp_remove
pcf857x_irq_domain_cleanup
pch_can_probe
pch_can_remove
pch_dma_probe
pch_dma_remove
pch_gbe_free_irq
pch_gpio_probe
pch_gpio_remove
pch_i2c_probe
pch_i2c_remove
pch_phub_probe
pch_phub_remove
pch_remove
pch_udc_remove
pch_vbus_gpio_free
pcie_pme_remove
pcim_iounmap
pci_oxsemi_tornado_init
pci_probe
pci_remove
pcmcia_disable_device
pcmcia_release_window
pcnet32_close
pcnet32_open
pcnet_close
pcxhr_free
pd6729_check_irq
pd6729_pci_probe
pd6729_pci_remove
peak_pci_probe
peak_pci_remove
phantom_probe
phantom_remove
phy_stop_interrupts
piix_disable_ahci
pixcir_i2c_ts_probe
```

```
pixcir_i2c_ts_remove
pluto2_probe
pluto2_remove
plx_pci_del_card
plx_pci_reset_marathon
pn544_hci_i2c_probe
pn544_hci_i2c_remove
ppp_shutdown_interface
prism54_remove
probe
pt1_probe
pt1_remove
pvscsi_release_resources
pvscsi_shutdown_intr
qib_6120_free_irq
qib_7220_free_irq
qib_7322_free_irq
qla1280_probe_one
qla1280_remove_one
qla25xx_free_rsp_que
qla4xxx_free_irqs
ql_adapter_down
ql_adapter_up
ql_free_irq
qt1070_probe
qt1070_remove
qt2160_probe
qt2160_remove
r6040_close
r6040_init_one
r6040_open
r6040_remove_one
r852_probe
r852_remove
radeon_device_fini
rc5t583_irq_exit
regmap_del_irq_chip
release_and_free_resource
release_card
release_cis_mem
release_io_space
release_memory_resource
release_resources
remove_inta_isr
reset_ivb_igd
rhine_init_one
rhine_remove_one
rio_probe1
rio_release_outb_dbell
rio_remove1
rr_init_one
rr_remove_one
rsxx_pci_probe
rsxx_pci_remove
rt2x00mmio_uninitialize
rtl8180_remove
rtl_pci_disconnect
rtl_pci_probe
rx8025_probe
rx8025_remove
s3_pci_probe
s3_pci_remove
saa7134_finidev
saa7134_initdev
saa7146_init_one
saa7146_remove_one
```



```
saa7164_finidev
sb1000_close
sbni_close
sc92031_probe
sc92031_remove
sca3000_probe
sca3000_remove
sdhci_add_host
sdhci_pci_remove_own_cd
sdhci_remove_host
sdhci_suspend_host
sdricoh_init_mmc
sep_probe
ser12_close
serial_hsu_remove
serial_unlink_irq_chain
setup_instance
sh_eth_close
sh_eth_open
si4713_probe
si4713_remove
sirdev_put_instance
sis900_probe
sis900_remove
sja1000_close
skfp_close
skge_remove
sky2_remove
slic_init_cleanup
smb347_remove
smsc_ircc_close
snd_ad1889_free
snd_ali_free
snd_als300_free
snd_atiixp_free
snd_azf3328_free
snd_bt87x_free
snd_ca0106_free
snd_cmipci_free
snd_cs4281_free
snd_cs46xx_free
snd_cs5535audio_free
snd_cx88_free
snd_echo_free
snd_emu10k1_free
snd_emu10k1x_free
snd_ensoniq_free
snd_es1938_free
snd_es1968_free
snd_fm801_free
snd_hdsp_free
snd_hdspm_free
snd_ice1712_free
snd_intel8x0_free
snd_intel8x0m_free
snd_korg1212_free
snd_lx6464es_create
snd_lx6464es_free
snd_m3_free
snd_mixart_free
snd_nm256_free
snd_nm256_release_irq
snd_riptide_free
snd_rme9652_free
snd_sbdsp_free
snd_sonicvibes_free
```

```
snd_trident_free
snd_uart16550_free
snd_via82xx_free
snd_vt1724_free
snd_vx222_free
snd_ymfpci_free
softing_card_shutdown
softing_enable_irq
ssb_iounmap
std_irq_cleanup
stmmac_dvr_remove
stmmac_open
stmmac_pci_remove
stmmac_release
stop_airo_card
stop_atmel_card
stop_ft1000_card
sundance_probe1
sundance_remove1
sym_free_resources
sym_iounmap_device
synaptics_i2c_remove
synaptics_rmi4_probe
synaptics_rmi4_remove
tifm_7xx1_probe
tifm_7xx1_remove
timbuart_shutdown
tlan_close
tpci200_unregister
tps65010_remove
tps6586x_i2c_probe
tps6586x_i2c_remove
tps65912_irq_exit
tsc2005_probe
tsi148_irq_exit
tsl2563_probe
tsl2563_remove
tsl2x7x_probe
tsl2x7x_remove
tulip_init_one
tulip_remove_one
__twa_shutdown
twl6030_init_irq
__twl_shutdown
__tw_shutdown
typhoon_close
typhoon_init_one
typhoon_open
typhoon_remove_one
typhoon_test_mmio
udc_pci_remove
uio_unregister_device
uli526x_init_one
uli526x_remove_one
ulite_shutdown
unbind_from_irqhandler
unload_mpu401
unload_pas
unload_uart6850
unregister_candev
unregister_networkdev
unregister_wlandev
usb_add_hcd
usb_remove_hcd
velocity_close
via_sd_remove
```



```
virtio_pci_probe
virtio_pci_remove
vlsi_close
vlsi_open
vmbus_bus_init
vmxnet3_free_irqs
vortex_close
vortex_init_one
vortex_open
vortex_remove_one
vt8623_pci_probe
vt8623_pci_remove
vxge_rem_isr
w83977af_close
w840_probe1
w840_remove1
wacom_i2c_probe
wacom_i2c_remove
wbsd_release_irq
whci_n_caps
whci_probe
whci_remove
wil6210_fini_irq
wil6210_request_3msi
wil_pcie_probe
wil_pcie_remove
wl3501_reset
wm2200_i2c_remove
wm5100_i2c_probe
wm5100_i2c_remove
wm831x_irq_exit
wm8350_irq_exit
wm8903_i2c_remove
wm8993_i2c_probe
wm8993_i2c_remove
xenvif_disconnect
xhci_try_enable_msi
xircom_probe
xircom_remove
xpc_destroy_gru_mq_uv
yam_close
yellowfin_init_one
yellowfin_remove_one
yenta_probe_cb_irq
```

```
**********************************************************
      41 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/no_return_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 __assert_fail
   2 exit
   3 __builtin_unreachable
```

```
**********************************************************
      82 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/no_return_funcs.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 main
   2 reserve_bootmem
   3 reserve_bootmem_node
   4 smp_reboot_interrupt
   5 xfs_fs_alloc_inode
```

```
**********************************************************
     218 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/smatch.common_functions
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 get_type
   2 sm_prefix
   3 add_hook
   4 strcmp
   5 snprintf
   6 get_filename
   7 get_argument_from_call_expr
   8 printf
   9 INT_PTR
  10 get_function
  11 free_string
  12 print_implied_debug_msg
  13 strip_expr
  14 get_lineno
  15 add_function_hook
  16 is_silenced_function
  17 fprintf
```

```
**********************************************************
       7 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/smatch_generic.common_functions
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 printf
```

**********
*1223 Fri Dec 21 15:00:24 2018*
*new/usr/src/tools/smatch/src/smatch_data/wine.bit_shifters*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
// list of macros used as shifters.
// generated by 'gen_bit_shifters.sh'
BANDID_FORMATBAR 3
BANDID_RULER 0
BANDID_STATUSBAR 1
BANDID_TOOLBAR 2
BWRITERVS_SWIZZLE_SHIFT 16
CANCEL_MSG_LINE 2
CERT_ALT_NAME_ENTRY_ERR_INDEX_SHIFT 16
CPU_ARM64 4
CPU_x86_64 1
D3DFVF_TEXCOUNT_SHIFT 8
D3DSI_INSTLENGTH_SHIFT 24
D3DSP_DCL_USAGEINDEX_SHIFT 16
D3DSP_DCL_USAGE_SHIFT 0
D3DSP_DSTSHIFT_SHIFT 24
D3DSP_REGTYPE_SHIFT 28
D3DSP_REGTYPE_SHIFT2 8
DIGIT_BIT 28
HASHTABLE_FLAG_BITS 6
INSTALLSTATE_LOCAL 3
JSSTR_LENGTH_SHIFT 4
MAX_TEXTURES 8
NET_WM_STATE_ABOVE 1
NET_WM_STATE_FULLSCREEN 0
NET_WM_STATE_MAXIMIZED 2
NET_WM_STATE_SKIP_PAGER 3
NET_WM_STATE_SKIP_TASKBAR 4
RESERVED_SHIFT 26
WINED3D_FFP_ATTRIBS_COUNT 15
WINED3D_FFP_BLENDINDICES 2
WINED3D_FFP_BLENDWEIGHT 1
WINED3D_FFP_DIFFUSE 5
WINED3D_FFP_NORMAL 3
WINED3D_FFP_POSITION 0
WINED3D_FFP_PSIZE 4
WINED3D_FFP_SPECULAR 6
WINED3D_SHADER_TYPE_COMPUTE 5
WINED3D_SHADER_TYPE_DOMAIN 4
WINED3D_SHADER_TYPE_GEOMETRY 2
WINED3D_SHADER_TYPE_HULL 3
WINED3D_SHADER_TYPE_PIXEL 0
WINED3D_SHADER_TYPE_VERTEX 1
WINED3D_SM1_REGISTER_TYPE_SHIFT 28
WINED3D_SM1_REGISTER_TYPE_SHIFT2 8
WINED3D_SM1_SWIZZLE_SHIFT 16
__WINE_DBCL_ERR 1
__WINE_DBCL_FIXME 0
__WINE_DBCL_INIT 7
__WINE_DBCL_TRACE 3
__WINE_DBCL_WARN 2
```

```
**********************************************************
      92 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/wine.ignored_macros
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 /*
2  * These macros are actively misleading to smatch so
3  * it's best to ignore them.
4  */
5 ok
```

```
**********************************************************
    1494 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/wine.no_return_funcs
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of functions which don't return.
// generated by 'gen_no_return_funcs.sh'
RaiseException
__assert_fail
exit
__builtin_unreachable
ExitProcess
_beginthread_trampoline
callback_exception
call_thread_func
close_socket_timeout
coff_process_info
collect_connections_proc
collect_query_thread
CorExitProcess
_CxxThrowException
debug_usage
DelayLoadFailureHook
do_sigsegv
do_usage
emfpathdrv_CreateDC
error
error_exit_fn
error_loc
error_loc_info
event_client
__ExceptionPtrRethrow
exit_on_signal
ExitProcess
exit_thread
fatal
fatal_error
fatal_perror
fatal_string_error
guiFatal
internal_error
iocp_poller
main
MSVCRT_abort
MSVCRT__wassert
nulldrv_CreateDC
nulldrv_DeleteDC
oob_client
oob_server
packet_kill
parser_error
pp_internal_error
ProvStore_releaseContext
PullPin_Thread_Main
raise_status
read_res16
REGPROC_print_error
relay_call_regs
RpcRaiseException
rpcrt4_ncacn_http_handoff
RtlExitUserProcess
__security_error_handler
segvhandler
select_server
server_connect_error
```



```
sigterm_callback
sigterm_handler
simple_client
simple_mixed_client
simple_server
start_process
start_thread
stub_entry_point
terminate_thread
test_thread_func_ex
textFatal
throw_bad_alloc
TIME_MMSysTimeThread
unwind_frame
unwind_target
user_error_fn
_vcomp_fork_worker
__widl_unwind_target
__wine_process_init
__wine_rtl_unwind
__wine_spec_exe_wentry
__wine_spec_unimplemented_stub
wine_switch_to_stack
WINMM_DevicesThreadProc
wmain
wpp_default_error
write_po_files
write_pot_file
wWinMain
xyyerror
yy_fatal_error
```

```
**********************************************************
      27 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/wine.no_return_funcs.add
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 ExitProcess
   2 RaiseException
```

```
**********************************************************
       6 Fri Dec 21 15:00:24 2018
new/usr/src/tools/smatch/src/smatch_data/wine.no_return_funcs.remove
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 usage
```

```
**********************************************************
    9977 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_data/wine.sizeof_param
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// list of function parameters that are the size of a buffer.
// generated by 'gen_sizeof_param.sh'
AddJobW 3 2
alloc_utf8_text 1 0
append 2 1
BCryptFinishHash 2 1
BCryptGenRandom 2 1
BCryptGetProperty 3 2
bsearch 3 1
call_minidriver 5 4
CertAddEncodedCertificateToStore 3 2
CertAddEncodedCRLToStore 3 2
CertAddEncodedCTLToStore 3 2
CertAddSerializedElementToStore 2 1
CertCreateCertificateContext 2 1
CertCreateCRLContext 2 1
CertCreateCTLContext 2 1
CertGetIntendedKeyUsage 3 2
CertNameToStrA 4 3
CertRDNValueToStrA 3 2
checkCRLHash 1 0
checkHash 1 0
check_index_buffer_ 4 2
check_param 4 3
CommitUrlCacheEntryW 6 5
compare_emf_bits 2 1
compare_file_data 2 1
compare_mf_bits 2 1
compare_mf_disk_bits 2 1
compareStore 3 2
CompareStringA 3 2
CompareStringA 5 2
convert_old_args 5 4
convert_str 1 4
copy_param 3 2
create_buffer_ 3 4
create_decoder 1 0
create_file 2 1
create_source_file 2 1
create_source_fileA 2 1
create_source_fileW 2 1
create_stream 1 0
create_stream_on_data 1 0
create_test_icon 6 5
CryptEncrypt 6 4
CryptEncryptMessage 4 3
CryptGenRandom 1 2
CryptGetMessageSignerCount 2 1
CryptHashCertificate 4 3
CryptHashData 2 1
CryptHashToBeSigned 3 2
CryptImportKey 2 1
CryptMsgUpdate 2 1
CryptVerifyDetachedMessageHash 2 1
CryptVerifyDetachedMessageSignature 3 2
CryptVerifyMessageHash 2 1
CryptVerifyMessageSignature 3 2
CryptVerifySignatureA 2 1
cstr_length 1 0
D3DXCreateCubeTextureFromFileInMemory 2 1
```



```
D3DXCreateCubeTextureFromFileInMemoryEx 2 1
D3DXCreateEffect 2 1
D3DXCreateEffectCompiler 1 0
D3DXCreateTextureFromFileInMemory 2 1
D3DXCreateTextureFromFileInMemoryEx 2 1
D3DXCreateVolumeTextureFromFileInMemory 2 1
D3DXGetImageInfoFromFileInMemory 1 0
D3DXLoadSurfaceFromFileInMemory 4 3
D3DXLoadSurfaceFromMemory 5 3
D3DXLoadVolumeFromFileInMemory 4 3
D3DXLoadVolumeFromMemory 6 3
dbg_read_memory 2 1
DdeCreateDataHandle 2 1
DeviceIoControl 3 2
DeviceIoControl 5 4
DIALOG_BrowsePrograms 2 1
DIALOG_BrowseSymbols 2 1
dns_ns_name_uncompress 4 3
dns_ns_name_unpack 4 3
doit 5 4
DragQueryFileA 3 2
DRIVER_GetLibName 3 2
ExpandEnvironmentStringsA 2 1
ExtEscape 2 3
ExtEscape 4 5
fgets 1 0
fill_sym_lvalue 4 3
format_exception_msg 2 1
format_hash 1 0
FormatMessageA 5 4
fread 1 0
fread 2 0
func_ptr 4 1
func_ptr 4 3
fwrite 1 0
fwrite 2 0
get_alsa_name_by_guid 2 1
GetAtomNameA 2 1
GetBitmapBits 1 2
get_buffer 1 0
get_builtin_path 3 2
GetClassNameA 2 1
GetClipboardFormatNameA 2 1
get_commands 3 2
get_config_key 4 3
GetCurrentDirectoryA 0 1
GetCurrentThemeName 1 0
GetCurrentThemeName 3 0
GetCurrentThemeName 5 4
GetDateFormatA 5 4
get_display_device_reg_key 1 0
GetDlgItemTextA 3 2
GetEnvironmentVariableA 2 1
GetFontData 4 3
GetFullPathNameA 1 2
GetGlyphOutlineA 4 5
GetGlyphOutlineW 4 5
gethostname 1 0
GetKeyNameTextA 2 1
get_line 1 0
GetLocaleInfoA 3 2
get_long_path_name 2 1
GetLongPathNameA 2 0
GetLongPathNameA 2 1
GetMetaFileBitsEx 1 2
GetModuleFileNameA 2 1
```

```
getnameinfo 3 2
getnameinfo 5 4
GetObjectA 1 2
GetObjectW 1 2
GetPrivateProfileSectionA 2 1
GetPrivateProfileStringA 4 3
get_process_info 2 1
GetProfileStringA 4 3
get_reg_value 4 3
GetRoleTextA 2 1
GetShortPathNameA 2 0
GetShortPathNameA 2 1
GetSystemDirectoryA 1 0
get_system_proxy_autoconfig_url 1 0
GetTempPathA 0 1
GetTextExtentExPointA 2 1
GetTextExtentPoint32A 2 1
GetTextFaceA 1 2
get_thread_info 3 2
GetTimeFormatA 5 4
GetTokenInformation 3 2
get_ttf_nametable_entry 3 2
GetUserObjectInformationA 3 2
GetUserObjectInformationW 3 2
GetWindowsDirectoryA 1 0
GetWindowTextA 2 1
GlobalGetAtomNameA 2 1
HCR_GetExecuteCommandW 4 3
HTTPREQ_Read 2 1
HttpSendRequestA 4 3
_hwrite 2 1
IContextMenu_GetCommandString 5 4
ID2D1Bitmap_CopyFromMemory 3 2
ID3D10Device_CreateGeometryShader 2 1
ID3D10Device_CreateInputLayout 4 3
ID3D10Device_CreatePixelShader 2 1
ID3D10Device_CreateVertexShader 2 1
ID3D10Device_UpdateSubresource 5 4
ID3D11DeviceContext_UpdateSubresource 5 4
ID3D11Device_CreateGeometryShader 2 1
ID3D11Device_CreateInputLayout 4 3
ID3D11Device_CreatePixelShader 2 1
ID3D11Device_CreateVertexShader 2 1
IDataConvert_DataConvert 3 5
IDataConvert_DataConvert 7 6
IDWriteGlyphRunAnalysis_CreateAlphaTexture 4 3
IHtmlLoadOptions_SetOption 3 2
IInternetProtocolEx_Read 2 1
IInternetProtocolInfo_QueryInfo 5 4
IInternetProtocol_Read 2 1
ILockBytes_ReadAt 3 2
ImmGetCompositionStringA 3 2
ImmSetCompositionStringA 3 2
ImmSetCompositionStringW 3 2
inet_ntop 3 2
initFileFromData 2 1
input_read_line 2 1
InternetReadFile 2 1
InternetSetOptionA 3 2
InternetTimeFromSystemTimeW 3 2
IoBuildDeviceIoControlRequest 5 4
IROTData_GetComparisonData 2 1
IsBadReadPtr 1 0
IsBadWritePtr 1 0
ISequentialStream_Read 2 1
IShellLinkA_GetArguments 2 1
```



```
IShellLinkA_GetDescription 2 1
IShellLinkA_GetIconLocation 2 1
IShellLinkA_GetPath 2 1
IShellLinkA_GetWorkingDirectory 2 1
IStream_Read 2 1
IStream_Write 2 1
IWICBitmapClipper_CopyPixels 3 4
IWICBitmap_CopyPixels 3 4
IWICBitmapFrameDecode_CopyPixels 3 4
IWICImagingFactory_CreateBitmapFromMemory 5 6
IWICStream_InitializeFromMemory 2 1
IWICStream_Write 2 1
LCMapStringA 5 2
LCMapStringA 5 4
load_entry 2 1
load_face 4 3
load_image 1 0
load_stream 2 1
LoadStringA 3 2
_lwrite 2 1
MCI_GetDevTypeFromFileName 2 1
mciGetErrorStringA 2 1
mciSendStringA 2 1
memcmp 2 0
memcmp 2 1
memcpy 2 0
memcpy 2 1
memory_get_string 5 4
memory_read_value 1 2
memset 2 0
mmioRead 2 1
MSFT_ReadLEWords 1 0
MSVCRT_asctime_s 1 0
MSVCRT__fwrite_nolock 1 0
MSVCRT__snwprintf 1 0
MultiByteToWideChar 3 2
NdrCorrelationInitialize 2 1
NetBTNameReq 5 4
NETCON_recv 2 1
NLS_RegEnumValue 3 2
NLS_RegEnumValue 5 4
NTDLL__vsnprintf 1 0
NtEnumerateKey 4 3
NtEnumerateValueKey 4 3
NtQueryDirectoryFile 6 5
NtQueryInformationProcess 3 2
NtQueryInformationThread 3 2
NtQueryKey 3 2
NtQueryValueKey 4 3
NtReadFile 6 5
NtSetInformationThread 3 2
NtSetValueKey 5 4
output 2 1
pBCryptHash 6 5
pCertNameToStrA 4 3
pCertRDNValueToStrA 3 2
pCertRDNValueToStrW 3 2
pCoInternetQueryInfo 4 3
pCryptDecodeObjectEx 3 2
pCryptFormatObject 6 5
pCryptVerifySignatureW 2 1
PeekNamedPipe 2 1
p_fgets 1 0
p_fread_s 1 0
pGetCalendarInfoA 4 3
pGetCalendarInfoW 4 3
```

```
pGetFileInformationByHandleEx 3 2
pGetFontFileData 4 3
pGetLongPathNameA 2 0
pGetLongPathNameA 2 1
pGetMappedFileNameA 3 2
pGetModuleBaseNameA 3 2
pGetModuleFileNameExA 3 2
pGetProcessImageFileNameA 2 1
pGetProcessImageFileNameW 2 1
pGetShortPathNameA 2 1
pGetSystemWow64DirectoryA 1 0
pGetVolumeNameForVolumeMountPointA 2 1
pGetVolumePathNamesForVolumeNameA 2 1
pGetVolumePathNamesForVolumeNameW 2 1
pGetWindowModuleFileNameA 2 1
pGetWsChanges 2 1
pHexFromBin 1 0
pIcmpSendEcho 3 2
pIdnToAscii 4 3
pIdnToUnicode 4 3
pInetNtop 3 2
pInternetGetConnectedStateExA 2 1
pInternetTimeFromSystemTimeA 3 2
pInternetTimeFromSystemTimeW 3 2
p_itoa_s 2 1
pK32GetProcessImageFileNameA 2 1
pLcidToRfc1766A 2 1
p_mbscat_s 1 0
p__mbscpy_s 1 0
p_mbslwr_s 1 0
p_mbsnbcat_s 1 0
p_mbsnbcat_s 3 2
p_mbsnbcpy_s 1 0
p_mbsnbcpy_s 3 2
p_mbsupr_s 1 0
pNtNotifyChangeDirectoryFile 6 5
pNtQueryDirectoryFile 6 5
pNtQueryInformationFile 3 2
pNtQueryInformationProcess 3 2
pNtQueryKey 3 2
pNtQueryLicenseValue 3 2
pNtQueryObject 3 2
pNtQueryVolumeInformationFile 3 2
pNtReadFile 6 5
pNtWriteFile 6 5
pPathUnExpandEnvStringsA 2 1
pQueryInformationJobObject 3 2
pReadDirectoryChangesW 2 1
pRegSetKeyValueW 5 4
printBytes 2 1
pRtlCompressBuffer 2 1
pRtlCompressBuffer 4 3
pRtlDecompressBuffer 2 1
pRtlDecompressFragment 2 1
pRtlIsTextUnicode 1 0
pRtlMultiByteToUnicodeN 1 0
pRtlUnicodeToUTF8N 1 0
pRtlUTF8ToUnicodeN 1 0
pSetupDiGetDeviceInstanceIdA 3 2
pSetupEnumInfSectionsA 3 2
pSetupGetFileCompressionInfoExA 2 1
pSHFormatDateTimeA 3 2
pSHGetIniStringW 3 2
pSHLWAPI_184 2 1
pSHLWAPI_212 2 1
pstrcat_s 1 0
```



```
pstrcpy_s 1 0
pstrerror_s 1 0
pStringTableLookUpStringEx 4 3
p_strlwr_s 1 0
p__strnset_s 1 0
p_ultoa_s 2 1
push_data 2 1
put_data 1 0
pXcvDataPort 5 4
qsort 2 0
QueryDosDeviceA 2 1
_read 2 1
read 2 1
read_bytes 1 0
read_bytes 2 1
ReadCharMetrics 2 1
read_data 2 1
ReadFile 2 1
ReadFileEx 2 1
ReadFontMetrics 2 1
read_func 2 1
ReadProcessMemory 3 2
read_stream 3 2
ReadString 2 1
RealGetWindowClassA 2 1
_receive_simple_request 3 2
recv 2 1
recvfrom 2 1
RegEnumKeyA 3 2
RegSetValueA 4 3
RegSetValueExA 5 4
RegSetValueExW 5 1
RegSetValueW 4 3
__res_query 4 3
reverse_lookup 2 1
rpcrt4_http_async_read 4 3
RtlGetCurrentDirectory_U 0 1
RtlMultiByteToUnicodeN 1 0
r_verify_reg_binary 5 4
SearchPathA 3 4
send 2 1
sendto 2 1
SetEnhMetaFileBits 0 1
SetMetaFileBitsEx 0 1
set_profile_device_key 2 1
SetupGetFileCompressionInfoExA 2 1
SetupGetLineTextA 5 4
SetupGetSourceFileLocationA 5 4
SetupGetSourceInfoA 4 3
SetupGetStringFieldA 3 2
SetupGetTargetPathA 4 3
setvbuf 3 1
SHELL_FindExecutableByVerb 4 3
snprintf 1 0
sock_recv 2 1
stab_strcpy 1 0
strftime 1 0
_Strftime 1 0
strncmp 2 0
strncmp 2 1
strncpyWtoA 2 0
sw_read_mem 3 2
SysAllocStringByteLen 1 0
SystemFunction036 1 0
test_add_certificate 1 0
test_buffer_object 2 1
```

```
391 test_font_metrics 5 4
392 _test_hkey_main_Value_A 3 2
393 _test_hkey_main_Value_W 3 2
394 test_LoadImageFile 2 1
395 test_LoadMeshFromX_ 9 7
396 test_moniker 3 2
397 test_moniker 5 4
398 test_moniker 7 6
399 test_output 3 2
400 test_persist_save_data 3 2
401 test_persist_save_data 5 2
402 test_persist_save_data 5 4
403 test_pic 1 0
404 test_preshader_op 2 1
405 unicode_expect_ 4 3
406 UpdateResourceA 5 4
407 UrlMkSetSessionOption 2 1
408 utf8_expect_ 4 3
409 VarTokenizeFormatString 2 1
410 vsnprintf 1 0
411 vsnprintfW 1 0
412 _vsprintf_p_wrapper 1 0
413 vsprintf_wrapper 2 1
414 vswprintf_wrapper 2 1
415 waveInAddBuffer 2 1
416 waveInGetErrorTextA 2 1
417 waveInPrepareHeader 2 1
418 waveInUnprepareHeader 2 1
419 waveOutGetErrorTextA 2 1
420 waveOutPrepareHeader 2 1
421 waveOutUnprepareHeader 2 1
422 waveOutWrite 2 1
423 WideCharToMultiByte 5 4
424 WINECON_GetConsoleTitle 2 1
425 wined3d_private_store_set_private_data 3 2
426 wine_dll_load 2 1
427 wine_dlopen 3 2
428 wine_dlsym 3 2
429 wine_init 3 2
430 wine_server_set_reply 2 1
431 WinHttpReadData 2 1
432 wld_read 2 1
433 _write 2 1
434 write 2 1
435 write_file 2 1
436 WriteFile 2 1
437 WriteFileEx 2 1
438 WriteProcessMemory 3 2
439 write_stream_data 3 2
440 WSAIoctl 3 2
441 WS_inet_ntop 3 2
442 WsWriteType 6 5
443 X11DRV_XDND_DescribeClipboardFormat 2 1
444 XLookupString 2 1
445 XmbLookupString 3 2
```

```
**********************************************************
    2571 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_data_source.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static char *get_source_parameter(struct expression *expr)
{
        struct expression *tmp;
        struct symbol *sym;
        char *name;
        int param;
        char *ret = NULL;
        char buf[32];
        int cnt = 0;

        tmp = expr;
        while ((tmp = get_assigned_expr(tmp))) {
                expr = tmp;
                if (cnt++ > 3)
                        break;
        }

        expr = strip_expr(expr);
        if (expr->type != EXPR_SYMBOL)
                return NULL;

        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                goto free;
        param = get_param_num_from_sym(sym);
        if (param < 0)
                goto free;
        if (param_was_set(expr))
                goto free;

        snprintf(buf, sizeof(buf), "p %d", param);
        ret = alloc_string(buf);

free:
        free_string(name);
        return ret;
}
```



```
static char *get_source_assignment(struct expression *expr)
{
        struct expression *right;
        char *name;
        char buf[64];
        char *ret;

        right = get_assigned_expr(expr);
        right = strip_expr(right);
        if (!right)
                return NULL;
        if (right->type != EXPR_CALL || right->fn->type != EXPR_SYMBOL)
                return NULL;
        if (is_fake_call(right))
                return NULL;
        name = expr_to_str(right->fn);
        if (!name)
                return NULL;
        snprintf(buf, sizeof(buf), "r %s", name);
        ret = alloc_string(buf);
        free_string(name);
        return ret;
}

static char *get_source_str(struct expression *arg)
{
        char *source;

        source = get_source_parameter(arg);
        if (source)
                return source;
        return get_source_assignment(arg);
}

static void match_caller_info(struct expression *expr)
{
        struct expression *arg;
        char *source;
        int i;

        i = -1;
        FOR_EACH_PTR(expr->args, arg) {
                i++;
                source = get_source_str(arg);
                if (!source)
                        continue;
                sql_insert_caller_info(expr, DATA_SOURCE, i, "$", source);
                free_string(source);
        } END_FOR_EACH_PTR(arg);
}

void register_data_source(int id)
{
//      if (!option_info)
//              return;
        my_id = id;
        add_hook(&match_caller_info, FUNCTION_CALL_HOOK);
}
```

```
**********************************************************
   64717 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_db.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <ctype.h>
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

struct sqlite3 *smatch_db;
struct sqlite3 *mem_db;
struct sqlite3 *cache_db;

static int return_id;

#define SQLITE_CACHE_PAGES 1000

struct def_callback {
        int hook_type;
        void (*callback)(const char *name, struct symbol *sym, char *key, char *
};
ALLOCATOR(def_callback, "definition db hook callbacks");
DECLARE_PTR_LIST(callback_list, struct def_callback);
static struct callback_list *select_caller_info_callbacks;

struct member_info_callback {
        int owner;
        void (*callback)(struct expression *call, int param, char *printed_name,
};
ALLOCATOR(member_info_callback, "caller_info callbacks");
DECLARE_PTR_LIST(member_info_cb_list, struct member_info_callback);
static struct member_info_cb_list *member_callbacks;

struct returned_state_callback {
        void (*callback)(int return_id, char *return_ranges, struct expression *
};
ALLOCATOR(returned_state_callback, "returned state callbacks");
DECLARE_PTR_LIST(returned_state_cb_list, struct returned_state_callback);
static struct returned_state_cb_list *returned_state_callbacks;

struct returned_member_callback {
        int owner;
        void (*callback)(int return_id, char *return_ranges, struct expression *
};
```



```
ALLOCATOR(returned_member_callback, "returned member callbacks");
DECLARE_PTR_LIST(returned_member_cb_list, struct returned_member_callback);
static struct returned_member_cb_list *returned_member_callbacks;

struct db_implies_callback {
        int type;
        void (*callback)(struct expression *call, struct expression *arg, char *
};
ALLOCATOR(db_implies_callback, "return_implies callbacks");
DECLARE_PTR_LIST(db_implies_cb_list, struct db_implies_callback);
static struct db_implies_cb_list *return_implies_cb_list;
static struct db_implies_cb_list *call_implies_cb_list;

/* silently truncates if needed. */
char *escape_newlines(const char *str)
{
        char buf[1024] = "";
        bool found = false;
        int i, j;

        for (i = 0, j = 0; str[i] != '\0' && j != sizeof(buf); i++, j++) {
                if (str[i] != '\n') {
                        buf[j] = str[i];
                        continue;
                }

                found = true;
                buf[j++] = '\\';
                if (j == sizeof(buf))
                        break;
                buf[j] = 'n';
        }

        if (!found)
                return alloc_sname(str);

        if (j == sizeof(buf))
                buf[j - 1] = '\0';
        return alloc_sname(buf);
}

static int print_sql_output(void *unused, int argc, char **argv, char **azColNam
{
        int i;

        for (i = 0; i < argc; i++) {
                if (i != 0)
                        printf(", ");
                sm_printf("%s", argv[i]);
        }
        sm_printf("\n");
        return 0;
}

void sql_exec(struct sqlite3 *db, int (*callback)(void*, int, char**, char**), v
{
        char *err = NULL;
        int rc;

        if (!db)
                return;

        if (option_debug) {
                sm_msg("%s", sql);
                if (strncasecmp(sql, "select", strlen("select")) == 0)
                        sqlite3_exec(db, sql, print_sql_output, NULL, NULL);
```

```
127         }

129         rc = sqlite3_exec(db, sql, callback, data, &err);
130         if (rc != SQLITE_OK && !parse_error) {
131                 sm_ierror("%s:%d SQL error #2: %s\n", get_filename(), get_lineno
132                 sm_ierror("%s:%d SQL: '%s'\n", get_filename(), get_lineno(), sql
133                 parse_error = 1;
134         }
135 }

137 static int replace_count;
138 static char **replace_table;
139 static const char *replace_return_ranges(const char *return_ranges)
140 {
141         int i;

143         if (!get_function()) {
144                 /* I have no idea why EXPORT_SYMBOL() is here */
145                 return return_ranges;
146         }
147         for (i = 0; i < replace_count; i += 3) {
148                 if (strcmp(replace_table[i + 0], get_function()) == 0) {
149                         if (strcmp(replace_table[i + 1], return_ranges) == 0)
150                                 return replace_table[i + 2];
151                 }
152         }
153         return return_ranges;
154 }


157 static char *use_states;
158 static int get_db_state_count(void)
159 {
160         struct sm_state *sm;
161         int count = 0;

163         FOR_EACH_SM(__get_cur_stree(), sm) {
164                 if (sm->owner == USHRT_MAX)
165                         continue;
166                 if (use_states[sm->owner])
167                         count++;
168         } END_FOR_EACH_SM(sm);
169         return count;
170 }

172 void db_ignore_states(int id)
173 {
174         use_states[id] = 0;
175 }

177 void sql_insert_return_states(int return_id, const char *return_ranges,
178                 int type, int param, const char *key, const char *value)
179 {
180         if (key && strlen(key) >= 80)
181                 return;
182         return_ranges = replace_return_ranges(return_ranges);
183         sql_insert(return_states, "'%s', '%s', %lu, %d, '%s', %d, %d, %d, '%s',
184                    get_base_file(), get_function(), (unsigned long)__inline_fn,
185                    return_id, return_ranges, fn_static(), type, param, key, valu
186 }

188 static struct string_list *common_funcs;
189 static int is_common_function(const char *fn)
190 {
191         char *tmp;
```



```
193         if (!fn)
194                 return 0;

196         if (strncmp(fn, "__builtin_", 10) == 0)
197                 return 1;

199         FOR_EACH_PTR(common_funcs, tmp) {
200                 if (strcmp(tmp, fn) == 0)
201                         return 1;
202         } END_FOR_EACH_PTR(tmp);

204         return 0;
205 }

207 static char *function_signature(void)
208 {
209         return type_to_str(get_real_base_type(cur_func_sym));
210 }

212 void sql_insert_caller_info(struct expression *call, int type,
213                 int param, const char *key, const char *value)
214 {
215         FILE *tmp_fd = sm_outfd;
216         char *fn;

218         if (!option_info && !__inline_call)
219                 return;

221         if (key && strlen(key) >= 80)
222                 return;

224         fn = get_fnptr_name(call->fn);
225         if (!fn)
226                 return;

228         if (__inline_call) {
229                 mem_sql(NULL, NULL,
230                         "insert into caller_info values ('%s', '%s', '%s', %lu,
231                         get_base_file(), get_function(), fn, (unsigned long)call
232                         is_static(call->fn), type, param, key, value);
233         }

235         if (!option_info)
236                 return;

238         if (strncmp(fn, "__builtin_", 10) == 0)
239                 return;
240         if (type != INTERNAL && is_common_function(fn))
241                 return;

243         sm_outfd = caller_info_fd;
244         sm_msg("SQL_caller_info: insert into caller_info values ("
245                "'%s', '%s', '%s', %%CALL_ID%%, %d, %d, %d, '%s', '%s');",
246                get_base_file(), get_function(), fn, is_static(call->fn),
247                type, param, key, value);
248         sm_outfd = tmp_fd;

250         free_string(fn);
251 }

253 void sql_insert_function_ptr(const char *fn, const char *struct_name)
254 {
255         sql_insert(function_ptr, "'%s', '%s', '%s', 0", get_base_file(), fn,
256                    struct_name);
257 }
```

```
259 void sql_insert_return_implies(int type, int param, const char *key, const char
260 {
261         sql_insert_or_ignore(return_implies, "'%s', '%s', %lu, %d, %d, %d, '%s',
262                 get_base_file(), get_function(), (unsigned long)__inline_fn,
263                 fn_static(), type, param, key, value);
264 }

266 void sql_insert_call_implies(int type, int param, const char *key, const char *v
267 {
268         sql_insert_or_ignore(call_implies, "'%s', '%s', %lu, %d, %d, %d, '%s', '
269                 get_base_file(), get_function(), (unsigned long)__inline_fn,
270                 fn_static(), type, param, key, value);
271 }

273 void sql_insert_function_type_size(const char *member, const char *ranges)
274 {
275         sql_insert(function_type_size, "'%s', '%s', '%s', '%s'", get_base_file()
276 }

278 void sql_insert_function_type_info(int type, const char *struct_type, const char
279 {
280         sql_insert(function_type_info, "'%s', '%s', %d, '%s', '%s', '%s'", get_b
281 }

283 void sql_insert_type_info(int type, const char *member, const char *value)
284 {
285         sql_insert_cache(type_info, "'%s', %d, '%s', '%s'", get_base_file(), typ
286 }

288 void sql_insert_local_values(const char *name, const char *value)
289 {
290         sql_insert(local_values, "'%s', '%s', '%s'", get_base_file(), name, valu
291 }

293 void sql_insert_function_type_value(const char *type, const char *value)
294 {
295         sql_insert(function_type_value, "'%s', '%s', '%s', '%s'", get_base_file(
296 }

298 void sql_insert_function_type(int param, const char *value)
299 {
300         sql_insert(function_type, "'%s', '%s', %d, %d, '%s'",
301                    get_base_file(), get_function(), fn_static(), param, value);
302 }

304 void sql_insert_parameter_name(int param, const char *value)
305 {
306         sql_insert(parameter_name, "'%s', '%s', %d, %d, '%s'",
307                    get_base_file(), get_function(), fn_static(), param, value);
308 }

310 void sql_insert_data_info(struct expression *data, int type, const char *value)
311 {
312         char *data_name;

314         data_name = get_data_info_name(data);
315         if (!data_name)
316                 return;
317         sql_insert(data_info, "'%s', '%s', %d, '%s'",
318                    is_static(data) ? get_base_file() : "extern",
319                    data_name, type, value);
320 }

322 void sql_insert_data_info_var_sym(const char *var, struct symbol *sym, int type,
323 {
324         sql_insert(data_info, "'%s', '%s', %d, '%s'",
```



```
325                    (sym->ctype.modifiers & MOD_STATIC) ? get_base_file() : "exte
326                    var, type, value);
327 }

329 void sql_save_constraint(const char *con)
330 {
331         if (!option_info)
332                 return;

334         sm_msg("SQL: insert or ignore into constraints (str) values('%s');", con
335 }

337 void sql_save_constraint_required(const char *data, int op, const char *limit)
338 {
339         sql_insert_or_ignore(constraints_required, "'%s', '%s', '%s'", data, sho
340 }

342 void sql_copy_constraint_required(const char *new_limit, const char *old_limit)
343 {
344         if (!option_info)
345                 return;

347         sm_msg("SQL_late: insert or ignore into constraints_required (data, op,
348                 "select constraints_required.data, constraints_required.op, '%s'
349                 "constraints_required where bound = '%s';", new_limit, old_limit
350 }

352 void sql_insert_fn_ptr_data_link(const char *ptr, const char *data)
353 {
354         sql_insert_or_ignore(fn_ptr_data_link, "'%s', '%s'", ptr, data);
355 }

357 void sql_insert_fn_data_link(struct expression *fn, int type, int param, const c
358 {
359         if (fn->type != EXPR_SYMBOL || !fn->symbol->ident)
360                 return;

362         sql_insert(fn_data_link, "'%s', '%s', %d, %d, %d, '%s', '%s'",
363                    (fn->symbol->ctype.modifiers & MOD_STATIC) ? get_base_file()
364                    fn->symbol->ident->name,
365                    !!(fn->symbol->ctype.modifiers & MOD_STATIC),
366                    type, param, key, value);
367 }

369 void sql_insert_mtag_about(mtag_t tag, const char *left_name, const char *right_
370 {
371         sql_insert(mtag_about, "%lld, '%s', '%s', %d, '%s', '%s'",
372                    tag, get_filename(), get_function(), get_lineno(), left_name,
373 }

375 void sql_insert_mtag_data(mtag_t tag, const char *var, int offset, int type, con
376 {
377         sql_insert(mtag_data, "%lld, '%s', %d, %d, '%s'", tag, var, offset, type
378 }

380 void sql_insert_mtag_map(mtag_t tag, int offset, mtag_t container)
381 {
382         sql_insert(mtag_map, "%lld, %d, %lld", tag, offset, container);
383 }

385 void sql_insert_mtag_alias(mtag_t orig, mtag_t alias)
386 {
387         sql_insert(mtag_alias, "%lld, %lld", orig, alias);
388 }

390 static int save_mtag(void *_tag, int argc, char **argv, char **azColName)
```

```
391 {
392         mtag_t *saved_tag = _tag;
393         mtag_t new_tag;

395         new_tag = strtoll(argv[0], NULL, 10);

397         if (!*saved_tag)
398                 *saved_tag = new_tag;
399         else if (*saved_tag != new_tag)
400                 *saved_tag = -1ULL;

402         return 0;
403 }

405 int mtag_map_select_container(mtag_t tag, int offset, mtag_t *container)
406 {
407         mtag_t tmp = 0;

409         run_sql(save_mtag, &tmp,
410                 "select container from mtag_map where tag = %lld and offset = %d
411                 tag, offset);

413         if (tmp == 0 || tmp == -1ULL)
414                 return 0;
415         *container = tmp;
416         return 1;
417 }

419 int mtag_map_select_tag(mtag_t container, int offset, mtag_t *tag)
420 {
421         mtag_t tmp = 0;

423         run_sql(save_mtag, &tmp,
424                 "select tag from mtag_map where container = %lld and offset = %d
425                 container, offset);

427         if (tmp == 0 || tmp == -1ULL)
428                 return 0;
429         *tag = tmp;
430         return 1;
431 }

433 char *get_static_filter(struct symbol *sym)
434 {
435         static char sql_filter[1024];

437         /* This can only happen on buggy code.  Return invalid SQL. */
438         if (!sym) {
439                 sql_filter[0] = '\0';
440                 return sql_filter;
441         }

443         if (sym->ctype.modifiers & MOD_STATIC) {
444                 snprintf(sql_filter, sizeof(sql_filter),
445                         "file = '%s' and function = '%s' and static = '1'",
446                         get_base_file(), sym->ident->name);
447         } else {
448                 snprintf(sql_filter, sizeof(sql_filter),
449                         "function = '%s' and static = '0'", sym->ident->name);
450         }

452         return sql_filter;
453 }

455 static int get_row_count(void *_row_count, int argc, char **argv, char **azColNa
456 {
```



```
457         int *row_count = _row_count;

459         *row_count = 0;
460         if (argc != 1)
461                 return 0;
462         *row_count = atoi(argv[0]);
463         return 0;
464 }

466 static void mark_call_params_untracked(struct expression *call)
467 {
468         struct expression *arg;
469         int i = 0;

471         FOR_EACH_PTR(call->args, arg) {
472                 mark_untracked(call, i++, "$", NULL);
473         } END_FOR_EACH_PTR(arg);
474 }

476 static void sql_select_return_states_pointer(const char *cols,
477         struct expression *call, int (*callback)(void*, int, char**, char**), vo
478 {
479         char *ptr;
480         int return_count = 0;

482         ptr = get_fnptr_name(call->fn);
483         if (!ptr)
484                 return;

486         run_sql(get_row_count, &return_count,
487                 "select count(*) from return_states join function_ptr "
488                 "where return_states.function == function_ptr.function and "
489                 "ptr = '%s' and searchable = 1 and type = %d;", ptr, INTERNAL);
490         /* The magic number 100 is just from testing on the kernel. */
491         if (return_count > 100) {
492                 mark_call_params_untracked(call);
493                 return;
494         }

496         run_sql(callback, info,
497                 "select %s from return_states join function_ptr where "
498                 "return_states.function == function_ptr.function and ptr = '%s'
499                 "and searchable = 1 "
500                 "order by function_ptr.file, return_states.file, return_id, type
501                 cols, ptr);
502 }

504 static int is_local_symbol(struct expression *expr)
505 {
506         if (expr->type != EXPR_SYMBOL)
507                 return 0;
508         if (expr->symbol->ctype.modifiers & (MOD_NONLOCAL | MOD_STATIC | MOD_ADD
509                 return 0;
510         return 1;
511 }

513 void sql_select_return_states(const char *cols, struct expression *call,
514         int (*callback)(void*, int, char**, char**), void *info)
515 {
516         int row_count = 0;

518         if (is_fake_call(call))
519                 return;

521         if (call->fn->type != EXPR_SYMBOL || !call->fn->symbol || is_local_symbo
522                 sql_select_return_states_pointer(cols, call, callback, info);
```

```
523                 return;
524         }

526         if (inlinable(call->fn)) {
527                 mem_sql(callback, info,
528                         "select %s from return_states where call_id = '%lu' orde
529                         cols, (unsigned long)call);
530                 return;
531         }

533         run_sql(get_row_count, &row_count, "select count(*) from return_states w
534                 get_static_filter(call->fn->symbol));
535         if (row_count > 3000)
536                 return;

538         run_sql(callback, info, "select %s from return_states where %s order by
539                 cols, get_static_filter(call->fn->symbol));
540 }

542 #define CALL_IMPLIES 0
543 #define RETURN_IMPLIES 1

545 struct implies_info {
546         int type;
547         struct db_implies_cb_list *cb_list;
548         struct expression *expr;
549         struct symbol *sym;
550 };

552 void sql_select_implies(const char *cols, struct implies_info *info,
553         int (*callback)(void*, int, char**, char**))
554 {
555         if (info->type == RETURN_IMPLIES && inlinable(info->expr->fn)) {
556                 mem_sql(callback, info,
557                         "select %s from return_implies where call_id = '%lu';",
558                         cols, (unsigned long)info->expr);
559                 return;
560         }

562         run_sql(callback, info, "select %s from %s_implies where %s;",
563                 cols,
564                 info->type == CALL_IMPLIES ? "call" : "return",
565                 get_static_filter(info->sym));
566 }

568 struct select_caller_info_data {
569         struct stree *final_states;
570         struct timeval start_time;
571         int prev_func_id;
572         int ignore;
573         int results;
574 };

576 static int caller_info_callback(void *_data, int argc, char **argv, char **azCol

578 static void sql_select_caller_info(struct select_caller_info_data *data,
579         const char *cols, struct symbol *sym)
580 {
581         if (__inline_fn) {
582                 mem_sql(caller_info_callback, data,
583                         "select %s from caller_info where call_id = %lu;",
584                         cols, (unsigned long)__inline_fn);
585                 return;
586         }

588         if (sym->ident->name && is_common_function(sym->ident->name))
```



```
589                 return;
590         run_sql(caller_info_callback, data,
591                 "select %s from common_caller_info where %s order by call_id;",
592                 cols, get_static_filter(sym));
593         if (data->results)
594                 return;

596         run_sql(caller_info_callback, data,
597                 "select %s from caller_info where %s order by call_id;",
598                 cols, get_static_filter(sym));
599 }

601 void select_caller_info_hook(void (*callback)(const char *name, struct symbol *s
602 {
603         struct def_callback *def_callback = __alloc_def_callback(0);

605         def_callback->hook_type = type;
606         def_callback->callback = callback;
607         add_ptr_list(&select_caller_info_callbacks, def_callback);
608 }

610 /*
611  * These call backs are used when the --info option is turned on to print struct
612  * member information.  For example foo->bar could have a state in
613  * smatch_extra.c and also check_user.c.
614  */
615 void add_member_info_callback(int owner, void (*callback)(struct expression *cal
616 {
617         struct member_info_callback *member_callback = __alloc_member_info_callb

619         member_callback->owner = owner;
620         member_callback->callback = callback;
621         add_ptr_list(&member_callbacks, member_callback);
622 }

624 void add_split_return_callback(void (*fn)(int return_id, char *return_ranges, st
625 {
626         struct returned_state_callback *callback = __alloc_returned_state_callba

628         callback->callback = fn;
629         add_ptr_list(&returned_state_callbacks, callback);
630 }

632 void add_returned_member_callback(int owner, void (*callback)(int return_id, cha
633 {
634         struct returned_member_callback *member_callback = __alloc_returned_memb

636         member_callback->owner = owner;
637         member_callback->callback = callback;
638         add_ptr_list(&returned_member_callbacks, member_callback);
639 }

641 void select_call_implies_hook(int type, void (*callback)(struct expression *call
642 {
643         struct db_implies_callback *cb = __alloc_db_implies_callback(0);

645         cb->type = type;
646         cb->callback = callback;
647         add_ptr_list(&call_implies_cb_list, cb);
648 }

650 void select_return_implies_hook(int type, void (*callback)(struct expression *ca
651 {
652         struct db_implies_callback *cb = __alloc_db_implies_callback(0);

654         cb->type = type;
```

```
655         cb->callback = callback;
656         add_ptr_list(&return_implies_cb_list, cb);
657 }

659 struct return_info {
660         struct expression *static_returns_call;
661         struct symbol *return_type;
662         struct range_list *return_range_list;
663 };

665 static int db_return_callback(void *_ret_info, int argc, char **argv, char **azC
666 {
667         struct return_info *ret_info = _ret_info;
668         struct range_list *rl;
669         struct expression *call_expr = ret_info->static_returns_call;

671         if (argc != 1)
672                 return 0;
673         call_results_to_rl(call_expr, ret_info->return_type, argv[0], &rl);
674         ret_info->return_range_list = rl_union(ret_info->return_range_list, rl);
675         return 0;
676 }

678 struct range_list *db_return_vals(struct expression *expr)
679 {
680         struct return_info ret_info = {};
681         char buf[64];
682         struct sm_state *sm;

684         if (is_fake_call(expr))
685                 return NULL;

687         snprintf(buf, sizeof(buf), "return %p", expr);
688         sm = get_sm_state(SMATCH_EXTRA, buf, NULL);
689         if (sm)
690                 return clone_rl(estate_rl(sm->state));
691         ret_info.static_returns_call = expr;
692         ret_info.return_type = get_type(expr);
693         if (!ret_info.return_type)
694                 return NULL;

696         if (expr->fn->type != EXPR_SYMBOL || !expr->fn->symbol)
697                 return NULL;

699         ret_info.return_range_list = NULL;
700         if (inlinable(expr->fn)) {
701                 mem_sql(db_return_callback, &ret_info,
702                         "select distinct return from return_states where call_id
703                         (unsigned long)expr);
704         } else {
705                 run_sql(db_return_callback, &ret_info,
706                         "select distinct return from return_states where %s;",
707                         get_static_filter(expr->fn->symbol));
708         }
709         return ret_info.return_range_list;
710 }

712 struct range_list *db_return_vals_from_str(const char *fn_name)
713 {
714         struct return_info ret_info;

716         ret_info.static_returns_call = NULL;
717         ret_info.return_type = &llong_ctype;
718         ret_info.return_range_list = NULL;

720         run_sql(db_return_callback, &ret_info,
```



```
721                 "select distinct return from return_states where function = '%s'
722                 fn_name);
723         return ret_info.return_range_list;
724 }

726 static void match_call_marker(struct expression *expr)
727 {
728         struct symbol *type;

730         type = get_type(expr->fn);
731         if (type && type->type == SYM_PTR)
732                 type = get_real_base_type(type);

734         /*
735          * we just want to record something in the database so that if we have
736          * two calls like:  frob(4); frob(some_unkown); then on the receiving
737          * side we know that sometimes frob is called with unknown parameters.
738          */

740         sql_insert_caller_info(expr, INTERNAL, -1, "%call_marker%", type_to_str(
741 }

743 static char *show_offset(int offset)
744 {
745         static char buf[64];

747         buf[0] = '\0';
748         if (offset != -1)
749                 snprintf(buf, sizeof(buf), "(-%d)", offset);
750         return buf;
751 }

753 static void print_struct_members(struct expression *call, struct expression *exp
754         void (*callback)(struct expression *call, int param, char *printed_name,
755 {
756         struct sm_state *sm;
757         char *name;
758         struct symbol *sym;
759         int len;
760         char printed_name[256];
761         int is_address = 0;
762         struct symbol *type;

764         expr = strip_expr(expr);
765         if (!expr)
766                 return;
767         if (expr->type == EXPR_PREOP && expr->op == '&') {
768                 expr = strip_expr(expr->unop);
769                 is_address = 1;
770         }

772         type = get_type(expr);
773         if (type && type_bits(type) < type_bits(&ulong_ctype))
774                 return;

776         name = expr_to_var_sym(expr, &sym);
777         if (!name || !sym)
778                 goto free;

780         len = strlen(name);
781         FOR_EACH_SM(stree, sm) {
782                 if (sm->sym != sym)
783                         continue;
784                 if (strcmp(name, sm->name) == 0) {
785                         if (is_address)
786                                 snprintf(printed_name, sizeof(printed_name), "*$
```

```
787                         else /* these are already handled. fixme: handle them he
788                                 continue;
789                 } else if (sm->name[0] == '*' && strcmp(name, sm->name + 1) == 0
790                         snprintf(printed_name, sizeof(printed_name), "*$%s", sho
791                 } else if (strncmp(name, sm->name, len) == 0) {
792                         if (isalnum(sm->name[len]))
793                                 continue;
794                         if (is_address)
795                                 snprintf(printed_name, sizeof(printed_name), "$%
796                         else
797                                 snprintf(printed_name, sizeof(printed_name), "$%
798                 } else {
799                         continue;
800                 }
801                 callback(call, param, printed_name, sm);
802         } END_FOR_EACH_SM(sm);
803 free:
804         free_string(name);
805 }
806
807 static int param_used_callback(void *_container, int argc, char **argv, char **a
808 {
809         char **container = _container;
810         static char buf[256];
811
812         snprintf(buf, sizeof(buf), "%s", argv[0]);
813         *container = buf;
814         return 0;
815 }
816
817 static void print_container_struct_members(struct expression *call, struct expre
818         void (*callback)(struct expression *call, int param, char *printed_name,
819 {
820         struct expression *tmp;
821         char *container = NULL;
822         int offset;
823         int holder_offset;
824         char *p;
825
826         if (!call->fn || call->fn->type != EXPR_SYMBOL || !call->fn->symbol)
827                 return;
828
829         /*
830          * We can't use the in-mem DB because we have to parse the function
831          * first, then we know if it takes a container, then we know to pass it
832          * the container data.
833          *
834          */
835         run_sql(&param_used_callback, &container,
836                 "select key from return_implies where %s and type = %d and key l
837                 get_static_filter(call->fn->symbol), CONTAINER, param);
838         if (!container)
839                 return;
840
841         p = strchr(container, '-');
842         if (!p)
843                 return;
844         offset = atoi(p);
845         p = strchr(p, ')');
846         if (!p)
847                 return;
848         p++;
849
850         tmp = get_assigned_expr(expr);
851         if (tmp)
852                 expr = tmp;
```



```
853
854         if (expr->type != EXPR_PREOP || expr->op != '&')
855                 return;
856         expr = strip_expr(expr->unop);
857         holder_offset = get_member_offset_from_deref(expr);
858         if (-holder_offset != offset)
859                 return;
860
861         expr = strip_expr(expr->deref);
862         if (expr->type == EXPR_PREOP && expr->op == '*')
863                 expr = strip_expr(expr->unop);
864
865         print_struct_members(call, expr, param, holder_offset, stree, callback);
866 }
867
868 static void match_call_info(struct expression *call)
869 {
870         struct member_info_callback *cb;
871         struct expression *arg;
872         struct stree *stree;
873         char *name;
874         int i;
875
876         name = get_fnptr_name(call->fn);
877         if (!name)
878                 return;
879
880         FOR_EACH_PTR(member_callbacks, cb) {
881                 stree = get_all_states_stree(cb->owner);
882                 i = 0;
883                 FOR_EACH_PTR(call->args, arg) {
884                         print_struct_members(call, arg, i, -1, stree, cb->callba
885                         print_container_struct_members(call, arg, i, stree, cb->
886                         i++;
887                 } END_FOR_EACH_PTR(arg);
888                 free_stree(&stree);
889         } END_FOR_EACH_PTR(cb);
890
891         free_string(name);
892 }
893
894 static int get_param(int param, char **name, struct symbol **sym)
895 {
896         struct symbol *arg;
897         int i;
898
899         i = 0;
900         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
901                 /*
902                  * this is a temporary hack to work around a bug (I think in spa
903                  * 2.6.37-rc1:fs/reiserfs/journal.o
904                  * If there is a function definition without parameter name foun
905                  * after a function implementation then it causes a crash.
906                  * int foo() {}
907                  * int bar(char *);
908                  */
909                 if (arg->ident->name < (char *)100)
910                         continue;
911                 if (i == param) {
912                         *name = arg->ident->name;
913                         *sym = arg;
914                         return TRUE;
915                 }
916                 i++;
917         } END_FOR_EACH_PTR(arg);
```

```
919         return FALSE;
920 }

922 static int function_signature_matches(const char *sig)
923 {
924         char *my_sig;

926         my_sig = function_signature();
927         if (!sig || !my_sig)
928                 return 1;  /* default to matching */
929         if (strcmp(my_sig, sig) == 0)
930                 return 1;
931         return 0;
932 }

934 static int caller_info_callback(void *_data, int argc, char **argv, char **azCol
935 {
936         struct select_caller_info_data *data = _data;
937         int func_id;
938         long type;
939         long param;
940         char *key;
941         char *value;
942         char *name = NULL;
943         struct symbol *sym = NULL;
944         struct def_callback *def_callback;
945         struct stree *stree;
946         struct timeval cur_time;

948         data->results = 1;

950         if (argc != 5)
951                 return 0;

953         gettimeofday(&cur_time, NULL);
954         if (cur_time.tv_sec - data->start_time.tv_sec > 10)
955                 return 0;

957         func_id = atoi(argv[0]);
958         errno = 0;
959         type = strtol(argv[1], NULL, 10);
960         param = strtol(argv[2], NULL, 10);
961         if (errno)
962                 return 0;
963         key = argv[3];
964         value = argv[4];

966         if (data->prev_func_id == -1)
967                 data->prev_func_id = func_id;
968         if (func_id != data->prev_func_id) {
969                 stree = __pop_fake_cur_stree();
970                 if (!data->ignore)
971                         merge_stree(&data->final_states, stree);
972                 free_stree(&stree);
973                 __push_fake_cur_stree();
974                 __unnullify_path();
975                 data->prev_func_id = func_id;
976                 data->ignore = 0;
977         }

979         if (data->ignore)
980                 return 0;
981         if (type == INTERNAL &&
982             !function_signature_matches(value)) {
983                 data->ignore = 1;
984                 return 0;
```



```
985         }

987         if (param >= 0 && !get_param(param, &name, &sym))
988                 return 0;

990         FOR_EACH_PTR(select_caller_info_callbacks, def_callback) {
991                 if (def_callback->hook_type == type)
992                         def_callback->callback(name, sym, key, value);
993         } END_FOR_EACH_PTR(def_callback);

995         return 0;
996 }

998 static struct string_list *ptr_names_done;
999 static struct string_list *ptr_names;

1001 static int get_ptr_name(void *unused, int argc, char **argv, char **azColName)
1002 {
1003         insert_string(&ptr_names, alloc_string(argv[0]));
1004         return 0;
1005 }

1007 static char *get_next_ptr_name(void)
1008 {
1009         char *ptr;

1011         FOR_EACH_PTR(ptr_names, ptr) {
1012                 if (list_has_string(ptr_names_done, ptr))
1013                         continue;
1014                 insert_string(&ptr_names_done, ptr);
1015                 return ptr;
1016         } END_FOR_EACH_PTR(ptr);
1017         return NULL;
1018 }

1020 static void get_ptr_names(const char *file, const char *name)
1021 {
1022         char sql_filter[1024];
1023         int before, after;

1025         if (file) {
1026                 snprintf(sql_filter, 1024, "file = '%s' and function = '%s';",
1027                          file, name);
1028         } else {
1029                 snprintf(sql_filter, 1024, "function = '%s';", name);
1030         }

1032         before = ptr_list_size((struct ptr_list *)ptr_names);

1034         run_sql(get_ptr_name, NULL,
1035                 "select distinct ptr from function_ptr where %s",
1036                 sql_filter);

1038         after = ptr_list_size((struct ptr_list *)ptr_names);
1039         if (before == after)
1040                 return;

1042         while ((name = get_next_ptr_name()))
1043                 get_ptr_names(NULL, name);
1044 }

1046 static void match_data_from_db(struct symbol *sym)
1047 {
1048         struct select_caller_info_data data = { .prev_func_id = -1 };
1049         struct sm_state *sm;
1050         struct stree *stree;
```

```
1051         struct timeval end_time;

1053         if (!sym || !sym->ident)
1054                 return;

1056         gettimeofday(&data.start_time, NULL);

1058         __push_fake_cur_stree();
1059         __unnullify_path();

1061         if (!__inline_fn) {
1062                 char *ptr;

1064                 if (sym->ctype.modifiers & MOD_STATIC)
1065                         get_ptr_names(get_base_file(), sym->ident->name);
1066                 else
1067                         get_ptr_names(NULL, sym->ident->name);

1069                 if (ptr_list_size((struct ptr_list *)ptr_names) > 20) {
1070                         __free_ptr_list((struct ptr_list **)&ptr_names);
1071                         __free_ptr_list((struct ptr_list **)&ptr_names_done);
1072                         stree = __pop_fake_cur_stree();
1073                         free_stree(&stree);
1074                         return;
1075                 }

1077                 sql_select_caller_info(&data,
1078                                        "call_id, type, parameter, key, value",
1079                                        sym);


1082                 stree = __pop_fake_cur_stree();
1083                 if (!data.ignore)
1084                         merge_stree(&data.final_states, stree);
1085                 free_stree(&stree);
1086                 __push_fake_cur_stree();
1087                 __unnullify_path();
1088                 data.prev_func_id = -1;
1089                 data.ignore = 0;

1091                 FOR_EACH_PTR(ptr_names, ptr) {
1092                         run_sql(caller_info_callback, &data,
1093                                 "select call_id, type, parameter, key, value"
1094                                 " from common_caller_info where function = '%s'
1095                                 ptr);
1096                 } END_FOR_EACH_PTR(ptr);

1098                 if (data.results) {
1099                         FOR_EACH_PTR(ptr_names, ptr) {
1100                                 free_string(ptr);
1101                         } END_FOR_EACH_PTR(ptr);
1102                         goto free_ptr_names;
1103                 }

1105                 FOR_EACH_PTR(ptr_names, ptr) {
1106                         run_sql(caller_info_callback, &data,
1107                                 "select call_id, type, parameter, key, value"
1108                                 " from caller_info where function = '%s' order b
1109                                 ptr);
1110                         free_string(ptr);
1111                 } END_FOR_EACH_PTR(ptr);

1113 free_ptr_names:
1114                 __free_ptr_list((struct ptr_list **)&ptr_names);
1115                 __free_ptr_list((struct ptr_list **)&ptr_names_done);
1116         } else {
```



```
1117                 sql_select_caller_info(&data,
1118                                        "call_id, type, parameter, key, value",
1119                                        sym);
1120         }

1122         stree = __pop_fake_cur_stree();
1123         if (!data.ignore)
1124                 merge_stree(&data.final_states, stree);
1125         free_stree(&stree);

1127         gettimeofday(&end_time, NULL);
1128         if (end_time.tv_sec - data.start_time.tv_sec <= 10) {
1129                 FOR_EACH_SM(data.final_states, sm) {
1130                         __set_sm(sm);
1131                 } END_FOR_EACH_SM(sm);
1132         }

1134         free_stree(&data.final_states);
1135 }

1137 static int return_implies_callbacks(void *_info, int argc, char **argv, char **a
1138 {
1139         struct implies_info *info = _info;
1140         struct db_implies_callback *cb;
1141         struct expression *arg = NULL;
1142         int type;
1143         int param;

1145         if (argc != 5)
1146                 return 0;

1148         type = atoi(argv[1]);
1149         param = atoi(argv[2]);

1151         FOR_EACH_PTR(info->cb_list, cb) {
1152                 if (cb->type != type)
1153                         continue;
1154                 if (param != -1) {
1155                         arg = get_argument_from_call_expr(info->expr->args, para
1156                         if (!arg)
1157                                 continue;
1158                 }
1159                 cb->callback(info->expr, arg, argv[3], argv[4]);
1160         } END_FOR_EACH_PTR(cb);

1162         return 0;
1163 }

1165 static int call_implies_callbacks(void *_info, int argc, char **argv, char **azC
1166 {
1167         struct implies_info *info = _info;
1168         struct db_implies_callback *cb;
1169         struct expression *arg;
1170         struct symbol *sym;
1171         char *name;
1172         int type;
1173         int param;

1175         if (argc != 5)
1176                 return 0;

1178         type = atoi(argv[1]);
1179         param = atoi(argv[2]);

1181         if (!get_param(param, &name, &sym))
1182                 return 0;
```

```
1183         arg = symbol_expression(sym);
1184         if (!arg)
1185                 return 0;

1187         FOR_EACH_PTR(info->cb_list, cb) {
1188                 if (cb->type != type)
1189                         continue;
1190                 cb->callback(info->expr, arg, argv[3], argv[4]);
1191         } END_FOR_EACH_PTR(cb);

1193         return 0;
1194 }

1196 static void match_return_implies(struct expression *expr)
1197 {
1198         struct implies_info info = {
1199                 .type = RETURN_IMPLIES,
1200                 .cb_list = return_implies_cb_list,
1201         };

1203         if (expr->fn->type != EXPR_SYMBOL ||
1204             !expr->fn->symbol)
1205                 return;
1206         info.expr = expr;
1207         info.sym = expr->fn->symbol;
1208         sql_select_implies("function, type, parameter, key, value", &info,
1209                            return_implies_callbacks);
1210 }

1212 static void match_call_implies(struct symbol *sym)
1213 {
1214         struct implies_info info = {
1215                 .type = CALL_IMPLIES,
1216                 .cb_list = call_implies_cb_list,
1217         };

1219         if (!sym || !sym->ident)
1220                 return;

1222         info.sym = sym;
1223         sql_select_implies("function, type, parameter, key, value", &info,
1224                            call_implies_callbacks);
1225 }

1227 static void print_initializer_list(struct expression_list *expr_list,
1228                 struct symbol *struct_type)
1229 {
1230         struct expression *expr;
1231         struct symbol *base_type;
1232         char struct_name[256];

1234         FOR_EACH_PTR(expr_list, expr) {
1235                 if (expr->type == EXPR_INDEX && expr->idx_expression && expr->id
1236                         print_initializer_list(expr->idx_expression->expr_list,
1237                         continue;
1238                 }
1239                 if (expr->type != EXPR_IDENTIFIER)
1240                         continue;
1241                 if (!expr->expr_ident)
1242                         continue;
1243                 if (!expr->ident_expression || !expr->ident_expression->symbol_n
1244                         continue;
1245                 base_type = get_type(expr->ident_expression);
1246                 if (!base_type || base_type->type != SYM_FN)
1247                         continue;
1248                 snprintf(struct_name, sizeof(struct_name), "(struct %s)->%s",
```



```
1249                         struct_type->ident->name, expr->expr_ident->name);
1250                 sql_insert_function_ptr(expr->ident_expression->symbol_name->nam
1251                                         struct_name);
1252         } END_FOR_EACH_PTR(expr);
1253 }

1255 static void global_variable(struct symbol *sym)
1256 {
1257         struct symbol *struct_type;

1259         if (!sym->ident)
1260                 return;
1261         if (!sym->initializer || sym->initializer->type != EXPR_INITIALIZER)
1262                 return;
1263         struct_type = get_base_type(sym);
1264         if (!struct_type)
1265                 return;
1266         if (struct_type->type == SYM_ARRAY) {
1267                 struct_type = get_base_type(struct_type);
1268                 if (!struct_type)
1269                         return;
1270         }
1271         if (struct_type->type != SYM_STRUCT || !struct_type->ident)
1272                 return;
1273         print_initializer_list(sym->initializer->expr_list, struct_type);
1274 }

1276 static void match_return_info(int return_id, char *return_ranges, struct express
1277 {
1278         sql_insert_return_states(return_id, return_ranges, INTERNAL, -1, "", fun
1279 }

1281 static void call_return_state_hooks_conditional(struct expression *expr)
1282 {
1283         struct returned_state_callback *cb;
1284         struct range_list *rl;
1285         char *return_ranges;
1286         int final_pass_orig = final_pass;

1288         __push_fake_cur_stree();

1290         final_pass = 0;
1291         __split_whole_condition(expr->conditional);
1292         final_pass = final_pass_orig;

1294         if (get_implied_rl(expr->cond_true, &rl))
1295                 rl = cast_rl(cur_func_return_type(), rl);
1296         else
1297                 rl = cast_rl(cur_func_return_type(), alloc_whole_rl(get_type(exp
1298         return_ranges = show_rl(rl);
1299         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(rl));

1301         return_id++;
1302         FOR_EACH_PTR(returned_state_callbacks, cb) {
1303                 cb->callback(return_id, return_ranges, expr->cond_true);
1304         } END_FOR_EACH_PTR(cb);

1306         __push_true_states();
1307         __use_false_states();

1309         if (get_implied_rl(expr->cond_false, &rl))
1310                 rl = cast_rl(cur_func_return_type(), rl);
1311         else
1312                 rl = cast_rl(cur_func_return_type(), alloc_whole_rl(get_type(exp
1313         return_ranges = show_rl(rl);
1314         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(rl));
```

```
1316         return_id++;
1317         FOR_EACH_PTR(returned_state_callbacks, cb) {
1318                 cb->callback(return_id, return_ranges, expr->cond_false);
1319         } END_FOR_EACH_PTR(cb);
1321         __merge_true_states();
1322         __free_fake_cur_stree();
1323 }
1325 static void call_return_state_hooks_compare(struct expression *expr)
1326 {
1327         struct returned_state_callback *cb;
1328         char *return_ranges;
1329         int final_pass_orig = final_pass;
1330         sval_t sval = { .type = &int_ctype };
1331         sval_t ret;
1333         if (!get_implied_value(expr, &ret))
1334                 ret.value = -1;
1336         __push_fake_cur_stree();
1338         final_pass = 0;
1339         __split_whole_condition(expr);
1340         final_pass = final_pass_orig;
1342         if (ret.value != 0) {
1343                 return_ranges = alloc_sname("1");
1344                 sval.value = 1;
1345                 set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_sval(sv
1347                 return_id++;
1348                 FOR_EACH_PTR(returned_state_callbacks, cb) {
1349                         cb->callback(return_id, return_ranges, expr);
1350                 } END_FOR_EACH_PTR(cb);
1351         }
1353         __push_true_states();
1354         __use_false_states();
1356         if (ret.value != 1) {
1357                 return_ranges = alloc_sname("0");
1358                 sval.value = 0;
1359                 set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_sval(sv
1361                 return_id++;
1362                 FOR_EACH_PTR(returned_state_callbacks, cb) {
1363                         cb->callback(return_id, return_ranges, expr);
1364                 } END_FOR_EACH_PTR(cb);
1365         }
1367         __merge_true_states();
1368         __free_fake_cur_stree();
1369 }
1371 static int ptr_in_list(struct sm_state *sm, struct state_list *slist)
1372 {
1373         struct sm_state *tmp;
1375         FOR_EACH_PTR(slist, tmp) {
1376                 if (strcmp(tmp->state->name, sm->state->name) == 0)
1377                         return 1;
1378         } END_FOR_EACH_PTR(tmp);
1380         return 0;
```



```
1381 }
1383 static char *get_return_compare_str(struct expression *expr)
1384 {
1385         char *compare_str;
1386         char *var;
1387         char buf[256];
1388         int comparison;
1389         int param;
1391         compare_str = expr_lte_to_param(expr, -1);
1392         if (compare_str)
1393                 return compare_str;
1394         param = get_param_num(expr);
1395         if (param < 0)
1396                 return NULL;
1398         var = expr_to_var(expr);
1399         if (!var)
1400                 return NULL;
1401         snprintf(buf, sizeof(buf), "%s orig", var);
1402         comparison = get_comparison_strings(var, buf);
1403         free_string(var);
1405         if (!comparison)
1406                 return NULL;
1408         snprintf(buf, sizeof(buf), "[%s$%d]", show_special(comparison), param);
1409         return alloc_sname(buf);
1410 }
1412 static int split_possible_helper(struct sm_state *sm, struct expression *expr)
1413 {
1414         struct returned_state_callback *cb;
1415         struct range_list *rl;
1416         char *return_ranges;
1417         struct sm_state *tmp;
1418         int ret = 0;
1419         int nr_possible, nr_states;
1420         char *compare_str = NULL;
1421         char buf[128];
1422         struct state_list *already_handled = NULL;
1424         if (!sm || !sm->merged)
1425                 return 0;
1427         if (too_many_possible(sm))
1428                 return 0;
1430         /* bail if it gets too complicated */
1431         nr_possible = ptr_list_size((struct ptr_list *)sm->possible);
1432         nr_states = get_db_state_count();
1433         if (nr_states * nr_possible >= 2000)
1434                 return 0;
1436         FOR_EACH_PTR(sm->possible, tmp) {
1437                 if (tmp->merged)
1438                         continue;
1439                 if (ptr_in_list(tmp, already_handled))
1440                         continue;
1441                 add_ptr_list(&already_handled, tmp);
1443                 ret = 1;
1444                 __push_fake_cur_stree();
1446                 overwrite_states_using_pool(sm, tmp);
```

```
1448                 rl = cast_rl(cur_func_return_type(), estate_rl(tmp->state));
1449                 return_ranges = show_rl(rl);
1450                 set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(clon
1451                 compare_str = get_return_compare_str(expr);
1452                 if (compare_str) {
1453                         snprintf(buf, sizeof(buf), "%s%s", return_ranges, compar
1454                         return_ranges = alloc_sname(buf);
1455                 }
1456
1457                 return_id++;
1458                 FOR_EACH_PTR(returned_state_callbacks, cb) {
1459                         cb->callback(return_id, return_ranges, expr);
1460                 } END_FOR_EACH_PTR(cb);
1461
1462                 __free_fake_cur_stree();
1463         } END_FOR_EACH_PTR(tmp);
1464
1465         free_slist(&already_handled);
1466
1467         return ret;
1468 }
1469
1470 static int call_return_state_hooks_split_possible(struct expression *expr)
1471 {
1472         struct sm_state *sm;
1473
1474         if (!expr || expr_equal_to_param(expr, -1))
1475                 return 0;
1476
1477         sm = get_sm_state_expr(SMATCH_EXTRA, expr);
1478         return split_possible_helper(sm, expr);
1479 }
1480
1481 static const char *get_return_ranges_str(struct expression *expr, struct range_l
1482 {
1483         struct range_list *rl;
1484         char *return_ranges;
1485         sval_t sval;
1486         char *compare_str;
1487         char *math_str;
1488         char buf[128];
1489
1490         *rl_p = NULL;
1491
1492         if (!expr)
1493                 return alloc_sname("");
1494
1495         if (get_implied_value(expr, &sval)) {
1496                 sval = sval_cast(cur_func_return_type(), sval);
1497                 *rl_p = alloc_rl(sval, sval);
1498                 return sval_to_str(sval);
1499         }
1500
1501         compare_str = expr_equal_to_param(expr, -1);
1502         math_str = get_value_in_terms_of_parameter_math(expr);
1503
1504         if (get_implied_rl(expr, &rl)) {
1505                 rl = cast_rl(cur_func_return_type(), rl);
1506                 return_ranges = show_rl(rl);
1507         } else if (get_imaginary_absolute(expr, &rl)){
1508                 rl = cast_rl(cur_func_return_type(), rl);
1509                 return alloc_sname(show_rl(rl));
1510         } else {
1511                 rl = cast_rl(cur_func_return_type(), alloc_whole_rl(get_type(exp
1512                 return_ranges = show_rl(rl);
```



```
1513         }
1514         *rl_p = rl;
1515
1516         if (compare_str) {
1517                 snprintf(buf, sizeof(buf), "%s%s", return_ranges, compare_str);
1518                 return alloc_sname(buf);
1519         }
1520         if (math_str) {
1521                 snprintf(buf, sizeof(buf), "%s[%s]", return_ranges, math_str);
1522                 return alloc_sname(buf);
1523         }
1524         compare_str = get_return_compare_str(expr);
1525         if (compare_str) {
1526                 snprintf(buf, sizeof(buf), "%s%s", return_ranges, compare_str);
1527                 return alloc_sname(buf);
1528         }
1529
1530         return return_ranges;
1531 }
1532
1533 static bool has_possible_negative(struct sm_state *sm)
1534 {
1535         struct sm_state *tmp;
1536
1537         FOR_EACH_PTR(sm->possible, tmp) {
1538                 if (!estate_rl(tmp->state))
1539                         continue;
1540                 if (sval_is_negative(estate_min(tmp->state)) &&
1541                     sval_is_negative(estate_max(tmp->state)))
1542                         return true;
1543         } END_FOR_EACH_PTR(tmp);
1544
1545         return false;
1546 }
1547
1548 static bool has_possible_zero_null(struct sm_state *sm)
1549 {
1550         struct sm_state *tmp;
1551         sval_t sval;
1552
1553         FOR_EACH_PTR(sm->possible, tmp) {
1554                 if (!estate_get_single_value(tmp->state, &sval))
1555                         continue;
1556                 if (sval.value == 0)
1557                         return true;
1558         } END_FOR_EACH_PTR(tmp);
1559
1560         return false;
1561 }
1562
1563 static int split_positive_from_negative(struct expression *expr)
1564 {
1565         struct sm_state *sm;
1566         struct returned_state_callback *cb;
1567         struct range_list *rl;
1568         const char *return_ranges;
1569         struct range_list *ret_rl;
1570         int undo;
1571
1572         /* We're going to print the states 3 times */
1573         if (get_db_state_count() > 10000 / 3)
1574                 return 0;
1575
1576         if (!get_implied_rl(expr, &rl) || !rl)
1577                 return 0;
1578         if (is_whole_rl(rl) || is_whole_rl_non_zero(rl))
```

```
1579                return 0;
1580        /* Forget about INT_MAX and larger */
1581        if (rl_max(rl).value <= 0)
1582                return 0;
1583        if (!sval_is_negative(rl_min(rl)))
1584                return 0;

1586        sm = get_sm_state_expr(SMATCH_EXTRA, expr);
1587        if (!sm)
1588                return 0;
1589        if (!has_possible_negative(sm))
1590                return 0;

1592        if (!assume(compare_expression(expr, '>', zero_expr())))
1593                return 0;

1595        return_id++;
1596        return_ranges = get_return_ranges_str(expr, &ret_rl);
1597        set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_rl));
1598        FOR_EACH_PTR(returned_state_callbacks, cb) {
1599                cb->callback(return_id, (char *)return_ranges, expr);
1600        } END_FOR_EACH_PTR(cb);

1602        end_assume();

1604        if (rl_has_sval(rl, sval_type_val(rl_type(rl), 0))) {
1605                undo = assume(compare_expression(expr, SPECIAL_EQUAL, zero_expr(

1607                return_id++;
1608                return_ranges = get_return_ranges_str(expr, &ret_rl);
1609                set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_
1610                FOR_EACH_PTR(returned_state_callbacks, cb) {
1611                        cb->callback(return_id, (char *)return_ranges, expr);
1612                } END_FOR_EACH_PTR(cb);

1614                if (undo)
1615                        end_assume();
1616        }

1618        undo = assume(compare_expression(expr, '<', zero_expr()));

1620        return_id++;
1621        return_ranges = get_return_ranges_str(expr, &ret_rl);
1622        set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_rl));
1623        FOR_EACH_PTR(returned_state_callbacks, cb) {
1624                cb->callback(return_id, (char *)return_ranges, expr);
1625        } END_FOR_EACH_PTR(cb);

1627        if (undo)
1628                end_assume();

1630        return 1;
1631 }

1633 static int call_return_state_hooks_split_null_non_null(struct expression *expr)
1634 {
1635        struct returned_state_callback *cb;
1636        struct range_list *rl;
1637        struct range_list *nonnull_rl;
1638        sval_t null_sval;
1639        struct range_list *null_rl = NULL;
1640        char *return_ranges;
1641        struct sm_state *sm;
1642        struct smatch_state *state;
1643        int nr_states;
1644        int final_pass_orig = final_pass;
```



```
1646        if (!expr || expr_equal_to_param(expr, -1))
1647                return 0;
1648        if (expr->type == EXPR_CALL)
1649                return 0;
1650        if (!is_pointer(expr))
1651                return 0;

1653        sm = get_sm_state_expr(SMATCH_EXTRA, expr);
1654        if (!sm)
1655                return 0;
1656        if (ptr_list_size((struct ptr_list *)sm->possible) == 1)
1657                return 0;
1658        state = sm->state;
1659        if (!estate_rl(state))
1660                return 0;
1661        if (estate_min(state).value == 0 && estate_max(state).value == 0)
1662                return 0;
1663        if (!has_possible_zero_null(sm))
1664                return 0;

1666        nr_states = get_db_state_count();
1667        if (option_info && nr_states >= 1500)
1668                return 0;

1670        rl = estate_rl(state);

1672        __push_fake_cur_stree();

1674        final_pass = 0;
1675        __split_whole_condition(expr);
1676        final_pass = final_pass_orig;

1678        nonnull_rl = rl_filter(rl, rl_zero());
1679        return_ranges = show_rl(nonnull_rl);
1680        set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(nonnull_rl));

1682        return_id++;
1683        FOR_EACH_PTR(returned_state_callbacks, cb) {
1684                cb->callback(return_id, return_ranges, expr);
1685        } END_FOR_EACH_PTR(cb);

1687        __push_true_states();
1688        __use_false_states();

1690        return_ranges = alloc_sname("0");
1691        null_sval = sval_type_val(rl_type(rl), 0);
1692        add_range(&null_rl, null_sval, null_sval);
1693        set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(null_rl));
1694        return_id++;
1695        FOR_EACH_PTR(returned_state_callbacks, cb) {
1696                cb->callback(return_id, return_ranges, expr);
1697        } END_FOR_EACH_PTR(cb);

1699        __merge_true_states();
1700        __free_fake_cur_stree();

1702        return 1;
1703 }

1705 static int call_return_state_hooks_split_success_fail(struct expression *expr)
1706 {
1707        struct sm_state *sm;
1708        struct range_list *rl;
1709        struct range_list *nonzero_rl;
1710        sval_t zero_sval;
```

```
1711         struct range_list *zero_rl = NULL;
1712         int nr_states;
1713         struct returned_state_callback *cb;
1714         char *return_ranges;
1715         int final_pass_orig = final_pass;

1717         if (option_project != PROJ_KERNEL)
1718                 return 0;

1720         nr_states = get_db_state_count();
1721         if (nr_states > 1500)
1722                 return 0;

1724         sm = get_sm_state_expr(SMATCH_EXTRA, expr);
1725         if (!sm)
1726                 return 0;
1727         if (ptr_list_size((struct ptr_list *)sm->possible) == 1)
1728                 return 0;

1730         rl = estate_rl(sm->state);
1731         if (!rl)
1732                 return 0;

1734         if (rl_min(rl).value < -4095 || rl_min(rl).value >= 0)
1735                 return 0;
1736         if (rl_max(rl).value != 0)
1737                 return 0;
1738         if (!has_possible_zero_null(sm))
1739                 return 0;

1741         __push_fake_cur_stree();

1743         final_pass = 0;
1744         __split_whole_condition(expr);
1745         final_pass = final_pass_orig;

1747         nonzero_rl = rl_filter(rl, rl_zero());
1748         nonzero_rl = cast_rl(cur_func_return_type(), nonzero_rl);
1749         return_ranges = show_rl(nonzero_rl);
1750         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(nonzero_rl));

1752         return_id++;
1753         FOR_EACH_PTR(returned_state_callbacks, cb) {
1754                 cb->callback(return_id, return_ranges, expr);
1755         } END_FOR_EACH_PTR(cb);

1757         __push_true_states();
1758         __use_false_states();

1760         return_ranges = alloc_sname("0");
1761         zero_sval = sval_type_val(rl_type(rl), 0);
1762         add_range(&zero_rl, zero_sval, zero_sval);
1763         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(zero_rl));
1764         return_id++;
1765         FOR_EACH_PTR(returned_state_callbacks, cb) {
1766                 cb->callback(return_id, return_ranges, expr);
1767         } END_FOR_EACH_PTR(cb);

1769         __merge_true_states();
1770         __free_fake_cur_stree();

1772         return 1;
1773 }

1775 static int is_boolean(struct expression *expr)
1776 {
```



```
1777         struct range_list *rl;

1779         if (!get_implied_rl(expr, &rl))
1780                 return 0;
1781         if (rl_min(rl).value == 0 && rl_max(rl).value == 1)
1782                 return 1;
1783         return 0;
1784 }

1786 static int is_conditional(struct expression *expr)
1787 {
1788         if (!expr)
1789                 return 0;
1790         if (expr->type == EXPR_CONDITIONAL || expr->type == EXPR_SELECT)
1791                 return 1;
1792         return 0;
1793 }

1795 static int splitable_function_call(struct expression *expr)
1796 {
1797         struct sm_state *sm;
1798         char buf[64];

1800         if (!expr || expr->type != EXPR_CALL)
1801                 return 0;
1802         snprintf(buf, sizeof(buf), "return %p", expr);
1803         sm = get_sm_state(SMATCH_EXTRA, buf, NULL);
1804         return split_possible_helper(sm, expr);
1805 }

1807 static struct sm_state *find_bool_param(void)
1808 {
1809         struct stree *start_states;
1810         struct symbol *arg;
1811         struct sm_state *sm, *tmp;
1812         sval_t sval;

1814         start_states = get_start_states();

1816         FOR_EACH_PTR_REVERSE(cur_func_sym->ctype.base_type->arguments, arg) {
1817                 if (!arg->ident)
1818                         continue;
1819                 sm = get_sm_state_stree(start_states, SMATCH_EXTRA, arg->ident->
1820                 if (!sm)
1821                         continue;
1822                 if (rl_min(estate_rl(sm->state)).value != 0 ||
1823                     rl_max(estate_rl(sm->state)).value != 1)
1824                         continue;
1825                 goto found;
1826         } END_FOR_EACH_PTR_REVERSE(arg);

1828         return NULL;

1830 found:
1831         /*
1832          * Check if it's splitable.  If not, then splitting it up is likely not
1833          * useful for the callers.
1834          */
1835         FOR_EACH_PTR(sm->possible, tmp) {
1836                 if (is_merged(tmp))
1837                         continue;
1838                 if (!estate_get_single_value(tmp->state, &sval))
1839                         return NULL;
1840         } END_FOR_EACH_PTR(tmp);

1842         return sm;
```

```
}

static int split_on_bool_sm(struct sm_state *sm, struct expression *expr)
{
        struct returned_state_callback *cb;
        struct range_list *ret_rl;
        const char *return_ranges;
        struct sm_state *tmp;
        int ret = 0;
        int nr_possible, nr_states;
        char *compare_str = NULL;
        char buf[128];
        struct state_list *already_handled = NULL;

        if (!sm || !sm->merged)
                return 0;

        if (too_many_possible(sm))
                return 0;

        /* bail if it gets too complicated */
        nr_possible = ptr_list_size((struct ptr_list *)sm->possible);
        nr_states = get_db_state_count();
        if (nr_states * nr_possible >= 2000)
                return 0;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (tmp->merged)
                        continue;
                if (ptr_in_list(tmp, already_handled))
                        continue;
                add_ptr_list(&already_handled, tmp);

                ret = 1;
                __push_fake_cur_stree();

                overwrite_states_using_pool(sm, tmp);

                return_ranges = get_return_ranges_str(expr, &ret_rl);
                set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_
                compare_str = get_return_compare_str(expr);
                if (compare_str) {
                        snprintf(buf, sizeof(buf), "%s%s", return_ranges, compar
                        return_ranges = alloc_sname(buf);
                }

                return_id++;
                FOR_EACH_PTR(returned_state_callbacks, cb) {
                        cb->callback(return_id, (char *)return_ranges, expr);
                } END_FOR_EACH_PTR(cb);

                __free_fake_cur_stree();
        } END_FOR_EACH_PTR(tmp);

        free_slist(&already_handled);

        return ret;
}

static int split_by_bool_param(struct expression *expr)
{
        struct sm_state *start_sm, *sm;
        sval_t sval;

        start_sm = find_bool_param();
        if (!start_sm)
```



```
                return 0;
        sm = get_sm_state(SMATCH_EXTRA, start_sm->name, start_sm->sym);
        if (!sm || estate_get_single_value(sm->state, &sval))
                return 0;
        return split_on_bool_sm(sm, expr);
}

static int split_by_null_nonnull_param(struct expression *expr)
{
        struct symbol *arg;
        struct sm_state *sm;
        sval_t zero = {
                .type = &ulong_ctype,
        };

        /* function must only take one pointer */
        if (ptr_list_size((struct ptr_list *)cur_func_sym->ctype.base_type->argu
                return 0;
        arg = first_ptr_list((struct ptr_list *)cur_func_sym->ctype.base_type->a
        if (!arg->ident)
                return 0;
        if (get_real_base_type(arg)->type != SYM_PTR)
                return 0;

        if (param_was_set_var_sym(arg->ident->name, arg))
                return 0;
        sm = get_sm_state(SMATCH_EXTRA, arg->ident->name, arg);
        if (!sm)
                return 0;

        if (!rl_has_sval(estate_rl(sm->state), zero))
                return 0;

        return split_on_bool_sm(sm, expr);
}

struct expression *strip_expr_statement(struct expression *expr)
{
        struct expression *orig = expr;
        struct statement *stmt, *last_stmt;

        if (!expr)
                return NULL;
        if (expr->type == EXPR_PREOP && expr->op == '(')
                expr = expr->unop;
        if (expr->type != EXPR_STATEMENT)
                return orig;
        stmt = expr->statement;
        if (!stmt || stmt->type != STMT_COMPOUND)
                return orig;

        last_stmt = last_ptr_list((struct ptr_list *)stmt->stmts);
        if (!last_stmt || last_stmt->type == STMT_LABEL)
                last_stmt = last_stmt->label_statement;
        if (!last_stmt || last_stmt->type != STMT_EXPRESSION)
                return orig;
        return strip_expr(last_stmt->expression);
}

static void call_return_state_hooks(struct expression *expr)
{
        struct returned_state_callback *cb;
        struct range_list *ret_rl;
        const char *return_ranges;
        int nr_states;
        sval_t sval;
```

```
1976         if (__path_is_null())
1977                 return;

1979         expr = strip_expr(expr);
1980         expr = strip_expr_statement(expr);

1982         if (is_impossible_path())
1983                 goto vanilla;

1985         if (expr && (expr->type == EXPR_COMPARE ||
1986                      !get_implied_value(expr, &sval)) &&
1987             (is_condition(expr) || is_boolean(expr))) {
1988                 call_return_state_hooks_compare(expr);
1989                 return;
1990         } else if (is_conditional(expr)) {
1991                 call_return_state_hooks_conditional(expr);
1992                 return;
1993         } else if (call_return_state_hooks_split_possible(expr)) {
1994                 return;
1995         } else if (call_return_state_hooks_split_null_non_null(expr)) {
1996                 return;
1997         } else if (call_return_state_hooks_split_success_fail(expr)) {
1998                 return;
1999         } else if (splitable_function_call(expr)) {
2000                 return;
2001         } else if (split_positive_from_negative(expr)) {
2002                 return;
2003         } else if (split_by_bool_param(expr)) {
2004         } else if (split_by_null_nonnull_param(expr)) {
2005                 return;
2006         }

2008 vanilla:
2009         return_ranges = get_return_ranges_str(expr, &ret_rl);
2010         set_state(RETURN_ID, "return_ranges", NULL, alloc_estate_rl(ret_rl));

2012         return_id++;
2013         nr_states = get_db_state_count();
2014         if (nr_states >= 10000) {
2015                 match_return_info(return_id, (char *)return_ranges, expr);
2016                 mark_all_params_untracked(return_id, (char *)return_ranges, expr
2017                 return;
2018         }
2019         FOR_EACH_PTR(returned_state_callbacks, cb) {
2020                 cb->callback(return_id, (char *)return_ranges, expr);
2021         } END_FOR_EACH_PTR(cb);
2022 }

2024 static void print_returned_struct_members(int return_id, char *return_ranges, st
2025 {
2026         struct returned_member_callback *cb;
2027         struct stree *stree;
2028         struct sm_state *sm;
2029         struct symbol *type;
2030         char *name;
2031         char member_name[256];
2032         int len;

2034         type = get_type(expr);
2035         if (!type || type->type != SYM_PTR)
2036                 return;
2037         name = expr_to_var(expr);
2038         if (!name)
2039                 return;
```



```
2041         member_name[sizeof(member_name) - 1] = '\0';
2042         strcpy(member_name, "$");

2044         len = strlen(name);
2045         FOR_EACH_PTR(returned_member_callbacks, cb) {
2046                 stree = __get_cur_stree();
2047                 FOR_EACH_MY_SM(cb->owner, stree, sm) {
2048                         if (sm->name[0] == '*' && strcmp(sm->name + 1, name) ==
2049                                 strcpy(member_name, "*$");
2050                                 cb->callback(return_id, return_ranges, expr, mem
2051                                 continue;
2052                         }
2053                         if (strncmp(sm->name, name, len) != 0)
2054                                 continue;
2055                         if (strncmp(sm->name + len, "->", 2) != 0)
2056                                 continue;
2057                         snprintf(member_name, sizeof(member_name), "$%s", sm->na
2058                         cb->callback(return_id, return_ranges, expr, member_name
2059                 } END_FOR_EACH_SM(sm);
2060         } END_FOR_EACH_PTR(cb);

2062         free_string(name);
2063 }

2065 static void reset_memdb(struct symbol *sym)
2066 {
2067         mem_sql(NULL, NULL, "delete from caller_info;");
2068         mem_sql(NULL, NULL, "delete from return_states;");
2069         mem_sql(NULL, NULL, "delete from call_implies;");
2070         mem_sql(NULL, NULL, "delete from return_implies;");
2071 }

2073 static void match_end_func_info(struct symbol *sym)
2074 {
2075         if (__path_is_null())
2076                 return;
2077         call_return_state_hooks(NULL);
2078 }

2080 static void match_after_func(struct symbol *sym)
2081 {
2082         if (!__inline_fn)
2083                 reset_memdb(sym);
2084 }

2086 static void init_memdb(void)
2087 {
2088         char *err = NULL;
2089         int rc;
2090         const char *schema_files[] = {
2091                 "db/db.schema",
2092                 "db/caller_info.schema",
2093                 "db/common_caller_info.schema",
2094                 "db/return_states.schema",
2095                 "db/function_type_size.schema",
2096                 "db/type_size.schema",
2097                 "db/function_type_info.schema",
2098                 "db/type_info.schema",
2099                 "db/call_implies.schema",
2100                 "db/return_implies.schema",
2101                 "db/function_ptr.schema",
2102                 "db/local_values.schema",
2103                 "db/function_type_value.schema",
2104                 "db/type_value.schema",
2105                 "db/function_type.schema",
2106                 "db/data_info.schema",
```

```
2107                "db/parameter_name.schema",
2108                "db/constraints.schema",
2109                "db/constraints_required.schema",
2110                "db/fn_ptr_data_link.schema",
2111                "db/fn_data_link.schema",
2112                "db/mtag_about.schema",
2113                "db/mtag_map.schema",
2114                "db/mtag_data.schema",
2115                "db/mtag_alias.schema",
2116        };
2117        static char buf[4096];
2118        int fd;
2119        int ret;
2120        int i;

2122        rc = sqlite3_open(":memory:", &mem_db);
2123        if (rc != SQLITE_OK) {
2124                sm_ierror("starting In-Memory database.");
2125                return;
2126        }

2128        for (i = 0; i < ARRAY_SIZE(schema_files); i++) {
2129                fd = open_schema_file(schema_files[i]);
2130                if (fd < 0)
2131                        continue;
2132                ret = read(fd, buf, sizeof(buf));
2133                if (ret < 0) {
2134                        sm_ierror("failed to read: %s", schema_files[i]);
2135                        continue;
2136                }
2137                close(fd);
2138                if (ret == sizeof(buf)) {
2139                        sm_ierror("Schema file too large:  %s (limit %zd bytes)"
2140                                schema_files[i], sizeof(buf));
2141                        continue;
2142                }
2143                buf[ret] = '\0';
2144                rc = sqlite3_exec(mem_db, buf, NULL, NULL, &err);
2145                if (rc != SQLITE_OK) {
2146                        sm_ierror("SQL error #2: %s", err);
2147                        sm_ierror("%s", buf);
2148                }
2149        }
2150 }

2152 static void init_cachedb(void)
2153 {
2154        char *err = NULL;
2155        int rc;
2156        const char *schema_files[] = {
2157                "db/call_implies.schema",
2158                "db/return_implies.schema",
2159                "db/type_info.schema",
2160                "db/mtag_data.schema",
2161                "db/sink_info.schema",
2162        };
2163        static char buf[4096];
2164        int fd;
2165        int ret;
2166        int i;

2168        rc = sqlite3_open(":memory:", &cache_db);
2169        if (rc != SQLITE_OK) {
2170                sm_ierror("starting In-Memory database.");
2171                return;
2172        }
```



```
2174        for (i = 0; i < ARRAY_SIZE(schema_files); i++) {
2175                fd = open_schema_file(schema_files[i]);
2176                if (fd < 0)
2177                        continue;
2178                ret = read(fd, buf, sizeof(buf));
2179                if (ret < 0) {
2180                        sm_ierror("failed to read: %s", schema_files[i]);
2181                        continue;
2182                }
2183                close(fd);
2184                if (ret == sizeof(buf)) {
2185                        sm_ierror("Schema file too large:  %s (limit %zd bytes)"
2186                                schema_files[i], sizeof(buf));
2187                        continue;
2188                }
2189                buf[ret] = '\0';
2190                rc = sqlite3_exec(cache_db, buf, NULL, NULL, &err);
2191                if (rc != SQLITE_OK) {
2192                        sm_ierror("SQL error #2: %s", err);
2193                        sm_ierror("%s", buf);
2194                }
2195        }
2196 }

2198 static int save_cache_data(void *_table, int argc, char **argv, char **azColName
2199 {
2200        static char buf[4096];
2201        char tmp[256];
2202        char *p = buf;
2203        char *table = _table;
2204        int i;


2207        p += snprintf(p, 4096 - (p - buf), "insert or ignore into %s values (",
2208        for (i = 0; i < argc; i++) {
2209                if (i)
2210                        p += snprintf(p, 4096 - (p - buf), ", ");
2211                sqlite3_snprintf(sizeof(tmp), tmp, "%q", argv[i]);
2212                p += snprintf(p, 4096 - (p - buf), "'%s'", tmp);

2214        }
2215        p += snprintf(p, 4096 - (p - buf), ");");
2216        if (p - buf > 4096)
2217                return 0;

2219        sm_msg("SQL: %s", buf);
2220        return 0;
2221 }

2223 static void dump_cache(struct symbol_list *sym_list)
2224 {
2225        if (!option_info)
2226                return;
2227        cache_sql(&save_cache_data, (char *)"type_info", "select * from type_inf
2228        cache_sql(&save_cache_data, (char *)"return_implies", "select * from ret
2229        cache_sql(&save_cache_data, (char *)"call_implies", "select * from call_
2230        cache_sql(&save_cache_data, (char *)"mtag_data", "select * from mtag_dat
2231        cache_sql(&save_cache_data, (char *)"sink_info", "select * from sink_inf
2232 }

2234 void open_smatch_db(char *db_file)
2235 {
2236        int rc;

2238        if (option_no_db)
```

```
2239                 return;

2241         use_states = malloc(num_checks + 1);
2242         memset(use_states, 0xff, num_checks + 1);

2244         init_memdb();
2245         init_cachedb();

2247         rc = sqlite3_open_v2(db_file, &smatch_db, SQLITE_OPEN_READONLY, NULL);
2248         if (rc != SQLITE_OK) {
2249                 option_no_db = 1;
2250                 return;
2251         }
2252         run_sql(NULL, NULL,
2253                 "PRAGMA cache_size = %d;", SQLITE_CACHE_PAGES);
2254         return;
2255 }

2257 static void register_common_funcs(void)
2258 {
2259         struct token *token;
2260         char *func;
2261         char filename[256];

2263         if (option_project == PROJ_NONE)
2264                 strcpy(filename, "common_functions");
2265         else
2266                 snprintf(filename, 256, "%s.common_functions", option_project_st

2268         token = get_tokens_file(filename);
2269         if (!token)
2270                 return;
2271         if (token_type(token) != TOKEN_STREAMBEGIN)
2272                 return;
2273         token = token->next;
2274         while (token_type(token) != TOKEN_STREAMEND) {
2275                 if (token_type(token) != TOKEN_IDENT)
2276                         return;
2277                 func = alloc_string(show_ident(token->ident));
2278                 add_ptr_list(&common_funcs, func);
2279                 token = token->next;
2280         }
2281         clear_token_alloc();
2282 }

2284 static char *get_next_string(char **str)
2285 {
2286         static char string[256];
2287         char *start;
2288         char *p = *str;
2289         int len;

2291         if (*p == '\0')
2292                 return NULL;
2293         start = p;

2295         while (*p != '\0' && *p != ' ' && *p != '\n')
2296                 p++;

2298         len = p - start;
2299         if (len > 256) {
2300                 memcpy(string, start, 255);
2301                 string[255] = '\0';
2302                 sm_ierror("return_fix: '%s' too long", string);
2303                 **str = '\0';
2304                 return NULL;
```



```
2305         }
2306         memcpy(string, start, len);
2307         string[len] = '\0';
2308         if (*p != '\0')
2309                 p++;
2310         *str = p;
2311         return string;
2312 }

2314 static void register_return_replacements(void)
2315 {
2316         char *func, *orig, *new;
2317         char filename[256];
2318         char buf[4096];
2319         int fd, ret, i;
2320         char *p;

2322         snprintf(filename, 256, "db/%s.return_fixes", option_project_str);
2323         fd = open_schema_file(filename);
2324         if (fd < 0)
2325                 return;
2326         ret = read(fd, buf, sizeof(buf));
2327         close(fd);
2328         if (ret < 0)
2329                 return;
2330         if (ret == sizeof(buf)) {
2331                 sm_ierror("file too large:  %s (limit %zd bytes)",
2332                         filename, sizeof(buf));
2333                 return;
2334         }
2335         buf[ret] = '\0';

2337         p = buf;
2338         while (*p) {
2339                 get_next_string(&p);
2340                 replace_count++;
2341         }
2342         if (replace_count == 0 || replace_count % 3 != 0) {
2343                 replace_count = 0;
2344                 return;
2345         }
2346         replace_table = malloc(replace_count * sizeof(char *));

2348         p = buf;
2349         i = 0;
2350         while (*p) {
2351                 func = alloc_string(get_next_string(&p));
2352                 orig = alloc_string(get_next_string(&p));
2353                 new  = alloc_string(get_next_string(&p));

2355                 replace_table[i++] = func;
2356                 replace_table[i++] = orig;
2357                 replace_table[i++] = new;
2358         }
2359 }

2361 void register_definition_db_callbacks(int id)
2362 {
2363         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
2364         add_hook(&global_variable, BASE_HOOK);
2365         add_hook(&global_variable, DECLARATION_HOOK);
2366         add_split_return_callback(match_return_info);
2367         add_split_return_callback(print_returned_struct_members);
2368         add_hook(&call_return_state_hooks, RETURN_HOOK);
2369         add_hook(&match_end_func_info, END_FUNC_HOOK);
2370         add_hook(&match_after_func, AFTER_FUNC_HOOK);
```

```
2372         add_hook(&match_data_from_db, FUNC_DEF_HOOK);
2373         add_hook(&match_call_implies, FUNC_DEF_HOOK);
2374         add_hook(&match_return_implies, CALL_HOOK_AFTER_INLINE);

2376         register_common_funcs();
2377         register_return_replacements();

2379         add_hook(&dump_cache, END_FILE_HOOK);
2380 }

2382 void register_db_call_marker(int id)
2383 {
2384         add_hook(&match_call_marker, FUNCTION_CALL_HOOK);
2385 }

2387 char *return_state_to_var_sym(struct expression *expr, int param, const char *ke
2388 {
2389         struct expression *arg;
2390         char *name = NULL;
2391         char member_name[256];

2393         *sym = NULL;

2395         if (param == -1) {
2396                 const char *star = "";

2398                 if (expr->type != EXPR_ASSIGNMENT)
2399                         return NULL;
2400                 name = expr_to_var_sym(expr->left, sym);
2401                 if (!name)
2402                         return NULL;
2403                 if (key[0] == '*') {
2404                         star = "*";
2405                         key++;
2406                 }
2407                 if (strncmp(key, "$", 1) != 0)
2408                         return name;
2409                 snprintf(member_name, sizeof(member_name), "%s%s%s", star, name,
2410                 free_string(name);
2411                 return alloc_string(member_name);
2412         }

2414         while (expr->type == EXPR_ASSIGNMENT)
2415                 expr = strip_expr(expr->right);
2416         if (expr->type != EXPR_CALL)
2417                 return NULL;

2419         arg = get_argument_from_call_expr(expr->args, param);
2420         if (!arg)
2421                 return NULL;

2423         return get_variable_from_key(arg, key, sym);
2424 }

2426 char *get_variable_from_key(struct expression *arg, const char *key, struct symb
2427 {
2428         char buf[256];
2429         char *tmp;

2431         if (!arg)
2432                 return NULL;

2434         arg = strip_expr(arg);

2436         if (strcmp(key, "$") == 0)
```



```
2437                 return expr_to_var_sym(arg, sym);

2439         if (strcmp(key, "*$") == 0) {
2440                 if (arg->type == EXPR_PREOP && arg->op == '&') {
2441                         arg = strip_expr(arg->unop);
2442                         return expr_to_var_sym(arg, sym);
2443                 } else {
2444                         tmp = expr_to_var_sym(arg, sym);
2445                         if (!tmp)
2446                                 return NULL;
2447                         snprintf(buf, sizeof(buf), "*%s", tmp);
2448                         free_string(tmp);
2449                         return alloc_string(buf);
2450                 }
2451         }

2453         if (arg->type == EXPR_PREOP && arg->op == '&') {
2454                 arg = strip_expr(arg->unop);
2455                 tmp = expr_to_var_sym(arg, sym);
2456                 if (!tmp)
2457                         return NULL;
2458                 snprintf(buf, sizeof(buf), "%s.%s", tmp, key + 3);
2459                 return alloc_string(buf);
2460         }

2462         tmp = expr_to_var_sym(arg, sym);
2463         if (!tmp)
2464                 return NULL;
2465         snprintf(buf, sizeof(buf), "%s%s", tmp, key + 1);
2466         free_string(tmp);
2467         return alloc_string(buf);
2468 }

2470 char *get_chunk_from_key(struct expression *arg, char *key, struct symbol **sym,
2471 {
2472         *vsl = NULL;

2474         if (strcmp("$", key) == 0)
2475                 return expr_to_chunk_sym_vsl(arg, sym, vsl);
2476         return get_variable_from_key(arg, key, sym);
2477 }

2479 const char *state_name_to_param_name(const char *state_name, const char *param_n
2480 {
2481         int name_len;
2482         static char buf[256];

2484         name_len = strlen(param_name);

2486         if (strcmp(state_name, param_name) == 0) {
2487                 return "$";
2488         } else if (state_name[name_len] == '-' && /* check for '-' from "->" */
2489             strncmp(state_name, param_name, name_len) == 0) {
2490                 snprintf(buf, sizeof(buf), "$%s", state_name + name_len);
2491                 return buf;
2492         } else if (state_name[0] == '*' && strcmp(state_name + 1, param_name) ==
2493                 return "*$";
2494         }
2495         return NULL;
2496 }

2498 const char *get_param_name_var_sym(const char *name, struct symbol *sym)
2499 {
2500         if (!sym || !sym->ident)
2501                 return NULL;
```

```
2503         return state_name_to_param_name(name, sym->ident->name);
2504 }

2506 const char *get_mtag_name_var_sym(const char *state_name, struct symbol *sym)
2507 {
2508         struct symbol *type;
2509         const char *sym_name;
2510         int name_len;
2511         static char buf[256];

2513         /*
2514          * mtag_name is different from param_name because mtags can be a struct
2515          * instead of a struct pointer.  But we want to treat it like a pointer
2516          * because really an mtag is a pointer.  Or in other words, if you pass
2517          * a struct foo then you want to talk about foo.bar but with an mtag
2518          * you want to refer to it as foo->bar.
2519          *
2520          */

2522         if (!sym || !sym->ident)
2523                 return NULL;

2525         type = get_real_base_type(sym);
2526         if (type && type->type == SYM_BASETYPE)
2527                 return "*$";

2529         sym_name = sym->ident->name;
2530         name_len = strlen(sym_name);

2532         if (state_name[name_len] == '.' && /* check for '-' from "->" */
2533             strncmp(state_name, sym_name, name_len) == 0) {
2534                 snprintf(buf, sizeof(buf), "$->%s", state_name + name_len + 1);
2535                 return buf;
2536         }

2538         return state_name_to_param_name(state_name, sym_name);
2539 }

2541 const char *get_mtag_name_expr(struct expression *expr)
2542 {
2543         char *name;
2544         struct symbol *sym;
2545         const char *ret = NULL;

2547         name = expr_to_var_sym(expr, &sym);
2548         if (!name || !sym)
2549                 goto free;

2551         ret = get_mtag_name_var_sym(name, sym);
2552 free:
2553         free_string(name);
2554         return ret;
2555 }

2557 const char *get_param_name(struct sm_state *sm)
2558 {
2559         return get_param_name_var_sym(sm->name, sm->sym);
2560 }

2562 char *get_data_info_name(struct expression *expr)
2563 {
2564         struct symbol *sym;
2565         char *name;
2566         char buf[256];
2567         char *ret = NULL;
```



```
2569         expr = strip_expr(expr);
2570         name = get_member_name(expr);
2571         if (name)
2572                 return name;
2573         name = expr_to_var_sym(expr, &sym);
2574         if (!name || !sym)
2575                 goto free;
2576         if (!(sym->ctype.modifiers & MOD_TOPLEVEL))
2577                 goto free;
2578         if (sym->ctype.modifiers & MOD_STATIC)
2579                 snprintf(buf, sizeof(buf), "static %s", name);
2580         else
2581                 snprintf(buf, sizeof(buf), "global %s", name);
2582         ret = alloc_sname(buf);
2583 free:
2584         free_string(name);
2585         return ret;
2586 }
```

```
**********************************************************
    6426 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_equiv.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * smatch_equiv.c is for tracking how variables are the same
 *
 * if (a == b) {
 * Or
 * x = y;
 *
 * When a variable gets modified all the old relationships are
 * deleted.  remove_equiv(expr);
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

ALLOCATOR(relation, "related variables");

static struct relation *alloc_relation(const char *name, struct symbol *sym)
{
        struct relation *tmp;

        tmp = __alloc_relation(0);
        tmp->name = alloc_string(name);
        tmp->sym = sym;
        return tmp;
}

struct related_list *clone_related_list(struct related_list *related)
{
        struct relation *rel;
        struct related_list *to_list = NULL;

        FOR_EACH_PTR(related, rel) {
                add_ptr_list(&to_list, rel);
        } END_FOR_EACH_PTR(rel);

        return to_list;
}

static int cmp_relation(struct relation *a, struct relation *b)
{
        int ret;
```



```
        if (a == b)
                return 0;

        if (a->sym > b->sym)
                return -1;
        if (a->sym < b->sym)
                return 1;

        ret = strcmp(a->name, b->name);
        if (ret)
                return ret;

        return 0;
}

struct related_list *get_shared_relations(struct related_list *one,
                                          struct related_list *two)
{
        struct related_list *ret = NULL;
        struct relation *one_rel;
        struct relation *two_rel;

        PREPARE_PTR_LIST(one, one_rel);
        PREPARE_PTR_LIST(two, two_rel);
        for (;;) {
                if (!one_rel || !two_rel)
                        break;
                if (cmp_relation(one_rel, two_rel) < 0) {
                        NEXT_PTR_LIST(one_rel);
                } else if (cmp_relation(one_rel, two_rel) == 0) {
                        add_ptr_list(&ret, one_rel);
                        NEXT_PTR_LIST(one_rel);
                        NEXT_PTR_LIST(two_rel);
                } else {
                        NEXT_PTR_LIST(two_rel);
                }
        }
        FINISH_PTR_LIST(two_rel);
        FINISH_PTR_LIST(one_rel);

        return ret;
}

static void debug_addition(struct related_list *rlist, const char *name)
{
        struct relation *tmp;

        if (!option_debug_related)
                return;

        sm_prefix();
        sm_printf("(");
        FOR_EACH_PTR(rlist, tmp) {
                sm_printf("%s ", tmp->name);
        } END_FOR_EACH_PTR(tmp);
        sm_printf(") <-- %s\n", name);
}

static void add_related(struct related_list **rlist, const char *name, struct sy
{
        struct relation *rel;
        struct relation *new;
        struct relation tmp = {
                .name = (char *)name,
                .sym = sym
```

```
127         };

129         debug_addition(*rlist, name);

131         FOR_EACH_PTR(*rlist, rel) {
132                 if (cmp_relation(rel, &tmp) < 0)
133                         continue;
134                 if (cmp_relation(rel, &tmp) == 0)
135                         return;
136                 new = alloc_relation(name, sym);
137                 INSERT_CURRENT(new, rel);
138                 return;
139         } END_FOR_EACH_PTR(rel);
140         new = alloc_relation(name, sym);
141         add_ptr_list(rlist, new);
142 }

144 static struct related_list *del_related(struct smatch_state *state, const char *
145 {
146         struct relation *tmp;
147         struct relation remove = {
148                 .name = (char *)name,
149                 .sym = sym,
150         };
151         struct related_list *ret = NULL;

153         FOR_EACH_PTR(estate_related(state), tmp) {
154                 if (cmp_relation(tmp, &remove) != 0)
155                         add_ptr_list(&ret, tmp);
156         } END_FOR_EACH_PTR(tmp);

158         return ret;
159 }

161 void remove_from_equiv(const char *name, struct symbol *sym)
162 {
163         struct sm_state *orig_sm;
164         struct relation *rel;
165         struct smatch_state *state;
166         struct related_list *to_update;

168         orig_sm = get_sm_state(SMATCH_EXTRA, name, sym);
169         if (!orig_sm || !get_dinfo(orig_sm->state)->related)
170                 return;

172         state = clone_estate(orig_sm->state);
173         to_update = del_related(state, name, sym);

175         FOR_EACH_PTR(to_update, rel) {
176                 struct sm_state *old_sm, *new_sm;

178                 old_sm = get_sm_state(SMATCH_EXTRA, rel->name, rel->sym);
179                 if (!old_sm)
180                         continue;

182                 new_sm = clone_sm(old_sm);
183                 get_dinfo(new_sm->state)->related = to_update;
184                 __set_sm(new_sm);
185         } END_FOR_EACH_PTR(rel);
186 }

188 void remove_from_equiv_expr(struct expression *expr)
189 {
190         char *name;
191         struct symbol *sym;
```



```
193         name = expr_to_var_sym(expr, &sym);
194         if (!name || !sym)
195                 goto free;
196         remove_from_equiv(name, sym);
197 free:
198         free_string(name);
199 }

201 void set_related(struct smatch_state *estate, struct related_list *rlist)
202 {
203         if (!estate_related(estate) && !rlist)
204                 return;
205         get_dinfo(estate)->related = rlist;
206 }

208 /*
209  * set_equiv() is only used for assignments where we set one variable
210  * equal to the other.  a = b;.  It's not used for if conditions where
211  * a == b.
212  */
213 void set_equiv(struct expression *left, struct expression *right)
214 {
215         struct sm_state *right_sm, *left_sm;
216         struct relation *rel;
217         char *left_name;
218         struct symbol *left_sym;
219         struct related_list *rlist;

221         left_name = expr_to_var_sym(left, &left_sym);
222         if (!left_name || !left_sym)
223                 goto free;

225         right_sm = get_sm_state_expr(SMATCH_EXTRA, right);
226         if (!right_sm)
227                 right_sm = set_state_expr(SMATCH_EXTRA, right, alloc_estate_whol
228         if (!right_sm)
229                 goto free;

231         /* This block is because we want to preserve the implications. */
232         left_sm = clone_sm(right_sm);
233         left_sm->name = alloc_string(left_name);
234         left_sm->sym = left_sym;
235         left_sm->state = clone_estate_cast(get_type(left), right_sm->state);
236         set_extra_mod_helper(left_name, left_sym, left, left_sm->state);
237         __set_sm(left_sm);

239         rlist = clone_related_list(estate_related(right_sm->state));
240         add_related(&rlist, right_sm->name, right_sm->sym);
241         add_related(&rlist, left_name, left_sym);

243         FOR_EACH_PTR(rlist, rel) {
244                 struct sm_state *old_sm, *new_sm;

246                 old_sm = get_sm_state(SMATCH_EXTRA, rel->name, rel->sym);
247                 if (!old_sm)  /* shouldn't happen */
248                         continue;
249                 new_sm = clone_sm(old_sm);
250                 new_sm->state = clone_estate(old_sm->state);
251                 get_dinfo(new_sm->state)->related = rlist;
252                 __set_sm(new_sm);
253         } END_FOR_EACH_PTR(rel);
254 free:
255         free_string(left_name);
256 }

258 void set_equiv_state_expr(int id, struct expression *expr, struct smatch_state *
```

```
259 {
260         struct relation *rel;
261         struct smatch_state *estate;

263         estate = get_state_expr(SMATCH_EXTRA, expr);

265         if (!estate)
266                 return;

268         FOR_EACH_PTR(get_dinfo(estate)->related, rel) {
269                 set_state(id, rel->name, rel->sym, state);
270         } END_FOR_EACH_PTR(rel);
271 }
```

```
**********************************************************
    9151 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_estate.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * smatch_dinfo.c has helper functions for handling data_info structs
 *
 */

#include <stdlib.h>
#ifndef __USE_ISOC99
#define __USE_ISOC99
#endif
#include <limits.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

struct smatch_state *merge_estates(struct smatch_state *s1, struct smatch_state
{
        struct smatch_state *tmp;
        struct range_list *value_ranges;
        struct related_list *rlist;

        if (estates_equiv(s1, s2))
                return s1;

        value_ranges = rl_union(estate_rl(s1), estate_rl(s2));
        tmp = alloc_estate_rl(value_ranges);
        rlist = get_shared_relations(estate_related(s1), estate_related(s2));
        set_related(tmp, rlist);
        if (estate_has_hard_max(s1) && estate_has_hard_max(s2))
                estate_set_hard_max(tmp);

        estate_set_fuzzy_max(tmp, sval_max(estate_get_fuzzy_max(s1), estate_get_

        return tmp;
}

struct data_info *get_dinfo(struct smatch_state *state)
{
        if (!state)
                return NULL;
        return (struct data_info *)state->data;
}
```



```
struct range_list *estate_rl(struct smatch_state *state)
{
        if (!state)
                return NULL;
        return get_dinfo(state)->value_ranges;
}

struct related_list *estate_related(struct smatch_state *state)
{
        if (!state)
                return NULL;
        return get_dinfo(state)->related;
}

sval_t estate_get_fuzzy_max(struct smatch_state *state)
{
        sval_t empty = {};

        if (!state || !get_dinfo(state))
                return empty;
        return get_dinfo(state)->fuzzy_max;
}

int estate_has_fuzzy_max(struct smatch_state *state)
{
        if (estate_get_fuzzy_max(state).type)
                return 1;
        return 0;
}

void estate_set_fuzzy_max(struct smatch_state *state, sval_t fuzzy_max)
{
        if (!rl_has_sval(estate_rl(state), fuzzy_max))
                return;
        get_dinfo(state)->fuzzy_max = fuzzy_max;
}

void estate_copy_fuzzy_max(struct smatch_state *new, struct smatch_state *old)
{
        if (!estate_has_fuzzy_max(old))
                return;
        estate_set_fuzzy_max(new, estate_get_fuzzy_max(old));
}

void estate_clear_fuzzy_max(struct smatch_state *state)
{
        sval_t empty = {};

        get_dinfo(state)->fuzzy_max = empty;
}

int estate_has_hard_max(struct smatch_state *state)
{
        if (!state)
                return 0;
        return get_dinfo(state)->hard_max;
}

void estate_set_hard_max(struct smatch_state *state)
{
        get_dinfo(state)->hard_max = 1;
}

void estate_clear_hard_max(struct smatch_state *state)
{
        get_dinfo(state)->hard_max = 0;
```

```
127 }

129 int estate_get_hard_max(struct smatch_state *state, sval_t *sval)
130 {
131         if (!state || !get_dinfo(state)->hard_max || !estate_rl(state))
132                 return 0;
133         *sval = rl_max(estate_rl(state));
134         return 1;
135 }

137 sval_t estate_min(struct smatch_state *state)
138 {
139         return rl_min(estate_rl(state));
140 }

142 sval_t estate_max(struct smatch_state *state)
143 {
144         return rl_max(estate_rl(state));
145 }

147 struct symbol *estate_type(struct smatch_state *state)
148 {
149         return rl_max(estate_rl(state)).type;
150 }

152 static int rlists_equiv(struct related_list *one, struct related_list *two)
153 {
154         struct relation *one_rel;
155         struct relation *two_rel;

157         PREPARE_PTR_LIST(one, one_rel);
158         PREPARE_PTR_LIST(two, two_rel);
159         for (;;) {
160                 if (!one_rel && !two_rel)
161                         return 1;
162                 if (!one_rel || !two_rel)
163                         return 0;
164                 if (one_rel->sym != two_rel->sym)
165                         return 0;
166                 if (strcmp(one_rel->name, two_rel->name))
167                         return 0;
168                 NEXT_PTR_LIST(one_rel);
169                 NEXT_PTR_LIST(two_rel);
170         }
171         FINISH_PTR_LIST(two_rel);
172         FINISH_PTR_LIST(one_rel);

174         return 1;
175 }

177 int estates_equiv(struct smatch_state *one, struct smatch_state *two)
178 {
179         if (!one || !two)
180                 return 0;
181         if (one == two)
182                 return 1;
183         if (!rlists_equiv(estate_related(one), estate_related(two)))
184                 return 0;
185         if (strcmp(one->name, two->name) == 0)
186                 return 1;
187         return 0;
188 }

190 int estate_is_whole(struct smatch_state *state)
191 {
192         return is_whole_rl(estate_rl(state));
```



```
193 }

195 int estate_is_empty(struct smatch_state *state)
196 {
197         return state && !estate_rl(state);
198 }

200 int estate_is_unknown(struct smatch_state *state)
201 {
202         if (!estate_is_whole(state))
203                 return 0;
204         if (estate_related(state))
205                 return 0;
206         if (estate_has_fuzzy_max(state))
207                 return 0;
208         return 1;
209 }

211 int estate_get_single_value(struct smatch_state *state, sval_t *sval)
212 {
213         sval_t min, max;

215         min = rl_min(estate_rl(state));
216         max = rl_max(estate_rl(state));
217         if (sval_cmp(min, max) != 0)
218                 return 0;
219         *sval = min;
220         return 1;
221 }

223 static struct data_info *alloc_dinfo(void)
224 {
225         struct data_info *ret;

227         ret = __alloc_data_info(0);
228         memset(ret, 0, sizeof(*ret));
229         return ret;
230 }

232 static struct data_info *alloc_dinfo_range(sval_t min, sval_t max)
233 {
234         struct data_info *ret;

236         ret = alloc_dinfo();
237         add_range(&ret->value_ranges, min, max);
238         return ret;
239 }

241 static struct data_info *alloc_dinfo_range_list(struct range_list *rl)
242 {
243         struct data_info *ret;

245         ret = alloc_dinfo();
246         ret->value_ranges = rl;
247         return ret;
248 }

250 static struct data_info *clone_dinfo(struct data_info *dinfo)
251 {
252         struct data_info *ret;

254         ret = alloc_dinfo();
255         ret->related = clone_related_list(dinfo->related);
256         ret->value_ranges = clone_rl(dinfo->value_ranges);
257         ret->hard_max = dinfo->hard_max;
258         ret->fuzzy_max = dinfo->fuzzy_max;
```

```
259         return ret;
260 }

262 struct smatch_state *clone_estate(struct smatch_state *state)
263 {
264         struct smatch_state *ret;

266         if (!state)
267                 return NULL;

269         ret = __alloc_smatch_state(0);
270         ret->name = state->name;
271         ret->data = clone_dinfo(get_dinfo(state));
272         return ret;
273 }

275 struct smatch_state *alloc_estate_empty(void)
276 {
277         struct smatch_state *state;
278         struct data_info *dinfo;

280         dinfo = alloc_dinfo();
281         state = __alloc_smatch_state(0);
282         state->data = dinfo;
283         state->name = "";
284         return state;
285 }

287 struct smatch_state *alloc_estate_whole(struct symbol *type)
288 {
289         return alloc_estate_rl(alloc_whole_rl(type));
290 }

292 struct smatch_state *extra_empty(void)
293 {
294         struct smatch_state *ret;

296         ret = __alloc_smatch_state(0);
297         ret->name = "empty";
298         ret->data = alloc_dinfo();
299         return ret;
300 }

302 struct smatch_state *alloc_estate_sval(sval_t sval)
303 {
304         struct smatch_state *state;

306         state = __alloc_smatch_state(0);
307         state->data = alloc_dinfo_range(sval, sval);
308         state->name = show_rl(get_dinfo(state)->value_ranges);
309         estate_set_hard_max(state);
310         estate_set_fuzzy_max(state, sval);
311         return state;
312 }

314 struct smatch_state *alloc_estate_range(sval_t min, sval_t max)
315 {
316         struct smatch_state *state;

318         state = __alloc_smatch_state(0);
319         state->data = alloc_dinfo_range(min, max);
320         state->name = show_rl(get_dinfo(state)->value_ranges);
321         return state;
322 }

324 struct smatch_state *alloc_estate_rl(struct range_list *rl)
```



```
325 {
326         struct smatch_state *state;

328         if (!rl)
329                 return extra_empty();

331         state = __alloc_smatch_state(0);
332         state->data = alloc_dinfo_range_list(rl);
333         state->name = show_rl(rl);
334         return state;
335 }

337 struct smatch_state *clone_estate_cast(struct symbol *type, struct smatch_state
338 {
339         struct smatch_state *ret;
340         struct data_info *dinfo;

342         if (!state)
343                 return NULL;

345         dinfo = alloc_dinfo();
346         dinfo->value_ranges = clone_rl(cast_rl(type, estate_rl(state)));

348         ret = __alloc_smatch_state(0);
349         ret->name = show_rl(dinfo->value_ranges);
350         ret->data = dinfo;

352         return ret;
353 }

355 struct smatch_state *get_implied_estate(struct expression *expr)
356 {
357         struct smatch_state *state;
358         struct range_list *rl;

360         state = get_state_expr(SMATCH_EXTRA, expr);
361         if (state)
362                 return state;
363         if (!get_implied_rl(expr, &rl))
364                 rl = alloc_whole_rl(get_type(expr));
365         return alloc_estate_rl(rl);
366 }

368 struct smatch_state *estate_filter_range(struct smatch_state *orig,
369                                          sval_t filter_min, sval_t filter_max)
370 {
371         struct range_list *rl;
372         struct smatch_state *state;

374         if (!orig)
375                 orig = alloc_estate_whole(filter_min.type);

377         rl = remove_range(estate_rl(orig), filter_min, filter_max);
378         state = alloc_estate_rl(rl);
379         if (estate_has_hard_max(orig))
380                 estate_set_hard_max(state);
381         if (estate_has_fuzzy_max(orig))
382                 estate_set_fuzzy_max(state, estate_get_fuzzy_max(orig));
383         return state;
384 }

386 struct smatch_state *estate_filter_sval(struct smatch_state *orig, sval_t sval)
387 {
388         return estate_filter_range(orig, sval, sval);
389 }
```

```
391 /*
392  * One of the complications is that smatch tries to free a bunch of data at the
393  * end of every function.
394  */
395 struct data_info *clone_dinfo_perm(struct data_info *dinfo)
396 {
397         struct data_info *ret;

399         ret = malloc(sizeof(*ret));
400         memset(ret, 0, sizeof(*ret));
401         ret->related = NULL;
402         ret->value_ranges = clone_rl_permanent(dinfo->value_ranges);
403         ret->hard_max = 0;
404         ret->fuzzy_max = dinfo->fuzzy_max;
405         return ret;
406 }

408 struct smatch_state *clone_estate_perm(struct smatch_state *state)
409 {
410         struct smatch_state *ret;

412         ret = malloc(sizeof(*ret));
413         ret->name = alloc_string(state->name);
414         ret->data = clone_dinfo_perm(get_dinfo(state));
415         return ret;
416 }
```

**********
*1356 Fri Dec 21 15:00:25 2018*
*new/usr/src/tools/smatch/src/smatch_expression_stacks.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_expression_stacks.h"

void push_expression(struct expression_list **estack, struct expression *expr)
{
        add_ptr_list(estack, expr);
}

struct expression *pop_expression(struct expression_list **estack)
{
        struct expression *expr;

        expr = last_ptr_list((struct ptr_list *)*estack);
        delete_ptr_list_last((struct ptr_list **)estack);
        return expr;
}

struct expression *top_expression(struct expression_list *estack)
{
        struct expression *expr;

        expr = last_ptr_list((struct ptr_list *)estack);
        return expr;
}

void free_expression_stack(struct expression_list **estack)
{
        __free_ptr_list((struct ptr_list **)estack);
}
```

```
**********************************************************
     276 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_expression_stacks.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 void push_expression(struct expression_list **estack, struct expression *expr);
2 struct expression *pop_expression(struct expression_list **estack);
3 struct expression *top_expression(struct expression_list *estack);
4 void free_expression_stack(struct expression_list **estack);
```

```
**********************************************************
    5107 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_expressions.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "smatch.h"
#include "smatch_extra.h"

DECLARE_ALLOCATOR(sname);
__ALLOCATOR(struct expression, "temporary expr", tmp_expression);

static struct position get_cur_pos(void)
{
        static struct position pos;
        static struct position none;
        struct expression *expr;
        struct statement *stmt;

        expr = last_ptr_list((struct ptr_list *)big_expression_stack);
        stmt = last_ptr_list((struct ptr_list *)big_statement_stack);
        if (expr)
                pos = expr->pos;
        else if (stmt)
                pos = stmt->pos;
        else
                pos = none;
        return pos;
}

struct expression *alloc_tmp_expression(struct position pos, int type)
{
        struct expression *expr;

        expr = __alloc_tmp_expression(0);
        expr->smatch_flags |= Fake;
        expr->type = type;
        expr->pos = pos;
        return expr;
}

void free_tmp_expressions(void)
{
        clear_tmp_expression_alloc();
}

struct expression *zero_expr(void)
{
        struct expression *zero;

        zero = alloc_tmp_expression(get_cur_pos(), EXPR_VALUE);
        zero->value = 0;
        zero->ctype = &int_ctype;
        return zero;
}

struct expression *value_expr(long long val)
{
        struct expression *expr;

        if (!val)
                return zero_expr();

        expr = alloc_tmp_expression(get_cur_pos(), EXPR_VALUE);
        expr->value = val;
        expr->ctype = &llong_ctype;
```



```
        return expr;
}

struct expression *member_expression(struct expression *deref, int op, struct id
{
        struct expression *expr;

        expr = alloc_tmp_expression(deref->pos, EXPR_DEREF);
        expr->op = op;
        expr->deref = deref;
        expr->member = member;
        expr->member_offset = -1;
        return expr;
}

struct expression *preop_expression(struct expression *expr, int op)
{
        struct expression *preop;

        preop = alloc_tmp_expression(expr->pos, EXPR_PREOP);
        preop->unop = expr;
        preop->op = op;
        return preop;
}

struct expression *deref_expression(struct expression *expr)
{
        return preop_expression(expr, '*');
}

struct expression *assign_expression(struct expression *left, int op, struct exp
{
        struct expression *expr;

        if (!right)
                return NULL;

        /* FIXME: make this a tmp expression. */
        expr = alloc_expression(right->pos, EXPR_ASSIGNMENT);
        expr->op = op;
        expr->left = left;
        expr->right = right;
        return expr;
}

struct expression *binop_expression(struct expression *left, int op, struct expr
{
        struct expression *expr;

        expr = alloc_tmp_expression(right->pos, EXPR_BINOP);
        expr->op = op;
        expr->left = left;
        expr->right = right;
        return expr;
}

struct expression *array_element_expression(struct expression *array, struct exp
{
        struct expression *expr;

        expr = binop_expression(array, '+', offset);
        return deref_expression(expr);
}

struct expression *symbol_expression(struct symbol *sym)
{
```

```
127         struct expression *expr;

129         expr = alloc_tmp_expression(sym->pos, EXPR_SYMBOL);
130         expr->symbol = sym;
131         expr->symbol_name = sym->ident;
132         return expr;
133 }

135 struct expression *compare_expression(struct expression *left, int op, struct ex
136 {
137         struct expression *expr;

139         expr = alloc_tmp_expression(get_cur_pos(), EXPR_COMPARE);
140         expr->op = op;
141         expr->left = left;
142         expr->right = right;
143         return expr;
144 }

146 struct expression *string_expression(char *str)
147 {
148         struct expression *ret;
149         struct string *string;
150         int len;

152         len = strlen(str) + 1;
153         string = (void *)__alloc_sname(4 + len);
154         string->length = len;
155         string->immutable = 0;
156         memcpy(string->data, str, len);

158         ret = alloc_tmp_expression(get_cur_pos(), EXPR_STRING);
159         ret->wide = 0;
160         ret->string = string;

162         return ret;
163 }

165 struct expression *gen_expression_from_key(struct expression *arg, const char *k
166 {
167         struct expression *ret;
168         struct token *token, *end;
169         const char *p = key;
170         char buf[4095];
171         char *alloc;
172         size_t len;

174         /* The idea is that we can parse either $0->foo or $->foo */
175         if (key[0] != '$')
176                 return NULL;
177         p++;
178         while (*p >= '0' && *p <= '9')
179                 p++;
180         len = snprintf(buf, sizeof(buf), "%s\n", p);
181         alloc = alloc_string(buf);

183         token = tokenize_buffer(alloc, len, &end);
184         if (!token)
185                 return NULL;
186         if (token_type(token) != TOKEN_STREAMBEGIN)
187                 return NULL;
188         token = token->next;

190         ret = arg;
191         while (token_type(token) == TOKEN_SPECIAL &&
192                token->special == SPECIAL_DEREFERENCE) {
```



```
193                 token = token->next;
194                 if (token_type(token) != TOKEN_IDENT)
195                         return NULL;
196                 ret = deref_expression(ret);
197                 ret = member_expression(ret, '*', token->ident);
198                 token = token->next;
199         }

201         if (token_type(token) != TOKEN_STREAMEND)
202                 return NULL;

204         return ret;
205 }

207 void expr_set_parent_expr(struct expression *expr, struct expression *parent)
208 {
209         if (!expr)
210                 return;
211         if (parent->smatch_flags & Fake)
212                 return;

214         expr->parent = (unsigned long)parent | 0x1UL;
215 }

217 void expr_set_parent_stmt(struct expression *expr, struct statement *parent)
218 {
219         if (!expr)
220                 return;
221         expr->parent = (unsigned long)parent;
222 }

224 struct expression *expr_get_parent_expr(struct expression *expr)
225 {
226         if (!expr)
227                 return NULL;
228         if (!(expr->parent & 0x1UL))
229                 return NULL;
230         return (struct expression *)(expr->parent & ~0x1UL);
231 }

233 struct statement *expr_get_parent_stmt(struct expression *expr)
234 {
235         if (!expr)
236                 return NULL;
237         if (expr->parent & 0x1UL)
238                 return NULL;
239         return (struct statement *)expr->parent;
240 }
```

```
**********************************************************
   68796 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_extra.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2008 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * smatch_extra.c is supposed to track the value of every variable.
 *
 */

#define _GNU_SOURCE
#include <string.h>

#include <stdlib.h>
#include <errno.h>
#ifndef __USE_ISOC99
#define __USE_ISOC99
#endif
#include <limits.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;
static int link_id;

static void match_link_modify(struct sm_state *sm, struct expression *mod_expr);

struct string_list *__ignored_macros = NULL;
static int in_warn_on_macro(void)
{
        struct statement *stmt;
        char *tmp;
        char *macro;

        stmt = get_current_statement();
        if (!stmt)
                return 0;
        macro = get_macro_name(stmt->pos);
        if (!macro)
                return 0;

        FOR_EACH_PTR(__ignored_macros, tmp) {
                if (!strcmp(tmp, macro))
                        return 1;
        } END_FOR_EACH_PTR(tmp);
        return 0;
```



```
}

typedef void (mod_hook)(const char *name, struct symbol *sym, struct expression
DECLARE_PTR_LIST(void_fn_list, mod_hook *);
static struct void_fn_list *extra_mod_hooks;
static struct void_fn_list *extra_nomod_hooks;

void add_extra_mod_hook(mod_hook *fn)
{
        mod_hook **p = malloc(sizeof(mod_hook *));
        *p = fn;
        add_ptr_list(&extra_mod_hooks, p);
}

void add_extra_nomod_hook(mod_hook *fn)
{
        mod_hook **p = malloc(sizeof(mod_hook *));
        *p = fn;
        add_ptr_list(&extra_nomod_hooks, p);
}

void call_extra_hooks(struct void_fn_list *hooks, const char *name, struct symbo
{
        mod_hook **fn;

        FOR_EACH_PTR(hooks, fn) {
                (*fn)(name, sym, expr, state);
        } END_FOR_EACH_PTR(fn);
}

void call_extra_mod_hooks(const char *name, struct symbol *sym, struct expressio
{
        call_extra_hooks(extra_mod_hooks, name, sym, expr, state);
}

void call_extra_nomod_hooks(const char *name, struct symbol *sym, struct express
{
        call_extra_hooks(extra_nomod_hooks, name, sym, expr, state);
}

static bool in_param_set;
void set_extra_mod_helper(const char *name, struct symbol *sym, struct expressio
{
        remove_from_equiv(name, sym);
        call_extra_mod_hooks(name, sym, expr, state);
        if ((__in_fake_assign || in_param_set) &&
            estate_is_unknown(state) && !get_state(SMATCH_EXTRA, name, sym))
                return;
        set_state(SMATCH_EXTRA, name, sym, state);
}

static void set_extra_nomod_helper(const char *name, struct symbol *sym, struct
{
        call_extra_nomod_hooks(name, sym, expr, state);
        set_state(SMATCH_EXTRA, name, sym, state);
}

static char *get_pointed_at(const char *name, struct symbol *sym, struct symbol
{
        struct expression *assigned;

        if (name[0] != '*')
                return NULL;
        if (strcmp(name + 1, sym->ident->name) != 0)
                return NULL;
```

```
127         assigned = get_assigned_expr_name_sym(sym->ident->name, sym);
128         if (!assigned)
129                 return NULL;
130         assigned = strip_parens(assigned);
131         if (assigned->type != EXPR_PREOP || assigned->op != '&')
132                 return NULL;
133
134         return expr_to_var_sym(assigned->unop, new_sym);
135 }
136
137 char *get_other_name_sym(const char *name, struct symbol *sym, struct symbol **n
138 {
139         struct expression *assigned;
140         char *orig_name = NULL;
141         char buf[256];
142         char *ret = NULL;
143         int skip;
144
145         *new_sym = NULL;
146
147         if (!sym || !sym->ident)
148                 return NULL;
149
150         ret = get_pointed_at(name, sym, new_sym);
151         if (ret)
152                 return ret;
153
154         skip = strlen(sym->ident->name);
155         if (name[skip] != '-' || name[skip + 1] != '>')
156                 return NULL;
157         skip += 2;
158
159         assigned = get_assigned_expr_name_sym(sym->ident->name, sym);
160         if (!assigned)
161                 return NULL;
162         if (assigned->type == EXPR_CALL)
163                 return map_call_to_other_name_sym(name, sym, new_sym);
164         if (assigned->type == EXPR_PREOP || assigned->op == '&') {
165
166                 orig_name = expr_to_var_sym(assigned, new_sym);
167                 if (!orig_name || !*new_sym)
168                         goto free;
169
170                 snprintf(buf, sizeof(buf), "%s.%s", orig_name + 1, name + skip);
171                 ret = alloc_string(buf);
172                 free_string(orig_name);
173                 return ret;
174         }
175
176         if (assigned->type != EXPR_DEREF)
177                 goto free;
178
179         orig_name = expr_to_var_sym(assigned, new_sym);
180         if (!orig_name || !*new_sym)
181                 goto free;
182
183         snprintf(buf, sizeof(buf), "%s->%s", orig_name, name + skip);
184         ret = alloc_string(buf);
185         free_string(orig_name);
186         return ret;
187
188 free:
189         free_string(orig_name);
190         return NULL;
191 }
```



```
193 void set_extra_mod(const char *name, struct symbol *sym, struct expression *expr
194 {
195         char *new_name;
196         struct symbol *new_sym;
197
198         set_extra_mod_helper(name, sym, expr, state);
199         new_name = get_other_name_sym(name, sym, &new_sym);
200         if (new_name && new_sym)
201                 set_extra_mod_helper(new_name, new_sym, expr, state);
202         free_string(new_name);
203 }
204
205 static struct expression *chunk_get_array_base(struct expression *expr)
206 {
207         /*
208          * The problem with is_array() is that it only returns true for things
209          * like foo[1] but not for foo[1].bar.
210          *
211          */
212         expr = strip_expr(expr);
213         while (expr && expr->type == EXPR_DEREF)
214                 expr = strip_expr(expr->deref);
215         return get_array_base(expr);
216 }
217
218 static int chunk_has_array(struct expression *expr)
219 {
220         return !!chunk_get_array_base(expr);
221 }
222
223 static void clear_array_states(struct expression *array)
224 {
225         struct sm_state *sm;
226
227         sm = get_sm_state_expr(link_id, array);
228         if (sm)
229                 match_link_modify(sm, NULL);
230 }
231
232 static void set_extra_array_mod(struct expression *expr, struct smatch_state *st
233 {
234         struct expression *array;
235         struct var_sym_list *vsl;
236         struct var_sym *vs;
237         char *name;
238         struct symbol *sym;
239
240         array = chunk_get_array_base(expr);
241
242         name = expr_to_chunk_sym_vsl(expr, &sym, &vsl);
243         if (!name || !vsl) {
244                 clear_array_states(array);
245                 goto free;
246         }
247
248         FOR_EACH_PTR(vsl, vs) {
249                 store_link(link_id, vs->var, vs->sym, name, sym);
250         } END_FOR_EACH_PTR(vs);
251
252         call_extra_mod_hooks(name, sym, expr, state);
253         set_state(SMATCH_EXTRA, name, sym, state);
254 free:
255         free_string(name);
256 }
257
258 void set_extra_expr_mod(struct expression *expr, struct smatch_state *state)
```

```
{
        struct symbol *sym;
        char *name;

        if (chunk_has_array(expr)) {
                set_extra_array_mod(expr, state);
                return;
        }

        expr = strip_expr(expr);
        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                goto free;
        set_extra_mod(name, sym, expr, state);
free:
        free_string(name);
}

void set_extra_nomod(const char *name, struct symbol *sym, struct expression *ex
{
        char *new_name;
        struct symbol *new_sym;
        struct relation *rel;
        struct smatch_state *orig_state;

        orig_state = get_state(SMATCH_EXTRA, name, sym);

        /* don't save unknown states if leaving it blank is the same */
        if (!orig_state && estate_is_unknown(state))
                return;

        new_name = get_other_name_sym(name, sym, &new_sym);
        if (new_name && new_sym)
                set_extra_nomod_helper(new_name, new_sym, expr, state);
        free_string(new_name);

        if (!estate_related(orig_state)) {
                set_extra_nomod_helper(name, sym, expr, state);
                return;
        }

        set_related(state, estate_related(orig_state));
        FOR_EACH_PTR(estate_related(orig_state), rel) {
                struct smatch_state *estate;

                if (option_debug_related)
                        sm_msg("%s updating related %s to %s", name, rel->name,
                estate = get_state(SMATCH_EXTRA, rel->name, rel->sym);
                if (!estate)
                        continue;
                set_extra_nomod_helper(rel->name, rel->sym, expr, clone_estate_c
        } END_FOR_EACH_PTR(rel);
}

void set_extra_nomod_vsl(const char *name, struct symbol *sym, struct var_sym_li
{
        struct var_sym *vs;

        FOR_EACH_PTR(vsl, vs) {
                store_link(link_id, vs->var, vs->sym, name, sym);
        } END_FOR_EACH_PTR(vs);

        set_extra_nomod(name, sym, expr, state);
}

/*
```



```
 * This is for return_implies_state() hooks which modify a SMATCH_EXTRA state
 */
void set_extra_expr_nomod(struct expression *expr, struct smatch_state *state)
{
        struct var_sym_list *vsl;
        struct var_sym *vs;
        char *name;
        struct symbol *sym;

        name = expr_to_chunk_sym_vsl(expr, &sym, &vsl);
        if (!name || !vsl)
                goto free;
        FOR_EACH_PTR(vsl, vs) {
                store_link(link_id, vs->var, vs->sym, name, sym);
        } END_FOR_EACH_PTR(vs);

        set_extra_nomod(name, sym, expr, state);
free:
        free_string(name);
}

static void set_extra_true_false(const char *name, struct symbol *sym,
                        struct smatch_state *true_state,
                        struct smatch_state *false_state)
{
        char *new_name;
        struct symbol *new_sym;
        struct relation *rel;
        struct smatch_state *orig_state;

        if (!true_state && !false_state)
                return;

        if (in_warn_on_macro())
                return;

        new_name = get_other_name_sym(name, sym, &new_sym);
        if (new_name && new_sym)
                set_true_false_states(SMATCH_EXTRA, new_name, new_sym, true_stat
        free_string(new_name);

        orig_state = get_state(SMATCH_EXTRA, name, sym);

        if (!estate_related(orig_state)) {
                set_true_false_states(SMATCH_EXTRA, name, sym, true_state, false
                return;
        }

        if (true_state)
                set_related(true_state, estate_related(orig_state));
        if (false_state)
                set_related(false_state, estate_related(orig_state));

        FOR_EACH_PTR(estate_related(orig_state), rel) {
                set_true_false_states(SMATCH_EXTRA, rel->name, rel->sym,
                                      true_state, false_state);
        } END_FOR_EACH_PTR(rel);
}

static void set_extra_chunk_true_false(struct expression *expr,
                                       struct smatch_state *true_state,
                                       struct smatch_state *false_state)
{
        struct var_sym_list *vsl;
        struct var_sym *vs;
        struct symbol *type;
```

```
391         char *name;
392         struct symbol *sym;

394         if (in_warn_on_macro())
395                 return;

397         type = get_type(expr);
398         if (!type)
399                 return;

401         name = expr_to_chunk_sym_vsl(expr, &sym, &vsl);
402         if (!name || !vsl)
403                 goto free;
404         FOR_EACH_PTR(vsl, vs) {
405                 store_link(link_id, vs->var, vs->sym, name, sym);
406         } END_FOR_EACH_PTR(vs);

408         set_true_false_states(SMATCH_EXTRA, name, sym,
409                               clone_estate(true_state),
410                               clone_estate(false_state));
411 free:
412         free_string(name);
413 }

415 static void set_extra_expr_true_false(struct expression *expr,
416                 struct smatch_state *true_state,
417                 struct smatch_state *false_state)
418 {
419         char *name;
420         struct symbol *sym;
421         sval_t sval;

423         if (!true_state && !false_state)
424                 return;

426         if (get_value(expr, &sval))
427                 return;

429         expr = strip_expr(expr);
430         name = expr_to_var_sym(expr, &sym);
431         if (!name || !sym) {
432                 free_string(name);
433                 set_extra_chunk_true_false(expr, true_state, false_state);
434                 return;
435         }
436         set_extra_true_false(name, sym, true_state, false_state);
437         free_string(name);
438 }

440 static int get_countdown_info(struct expression *condition, struct expression **
441 {
442         struct expression *unop_expr;
443         int comparison;
444         sval_t limit;

446         right->type = &int_ctype;
447         right->value = 0;

449         condition = strip_expr(condition);

451         if (condition->type == EXPR_COMPARE) {
452                 comparison = remove_unsigned_from_comparison(condition->op);

454                 if (comparison != SPECIAL_GTE && comparison != '>')
455                         return 0;
456                 if (!get_value(condition->right, &limit))
```



```
457                         return 0;

459                 unop_expr = condition->left;
460                 if (unop_expr->type != EXPR_PREOP && unop_expr->type != EXPR_POS
461                         return 0;
462                 if (unop_expr->op != SPECIAL_DECREMENT)
463                         return 0;

465                 *unop = unop_expr;
466                 *op = comparison;
467                 *right = limit;

469                 return 1;
470         }

472         if (condition->type != EXPR_PREOP && condition->type != EXPR_POSTOP)
473                 return 0;
474         if (condition->op != SPECIAL_DECREMENT)
475                 return 0;

477         *unop = condition;
478         *op = '>';

480         return 1;
481 }

483 static struct sm_state *handle_canonical_while_count_down(struct statement *loop
484 {
485         struct expression *iter_var;
486         struct expression *condition, *unop;
487         struct sm_state *sm;
488         struct smatch_state *estate;
489         int op;
490         sval_t start, right;

492         right.type = &int_ctype;
493         right.value = 0;

495         condition = strip_expr(loop->iterator_pre_condition);
496         if (!condition)
497                 return NULL;

499         if (!get_countdown_info(condition, &unop, &op, &right))
500                 return NULL;

502         iter_var = unop->unop;

504         sm = get_sm_state_expr(SMATCH_EXTRA, iter_var);
505         if (!sm)
506                 return NULL;
507         if (sval_cmp(estate_min(sm->state), right) < 0)
508                 return NULL;
509         start = estate_max(sm->state);
510         if  (sval_cmp(start, right) <= 0)
511                 return NULL;
512         if (!sval_is_max(start))
513                 start.value--;

515         if (op == SPECIAL_GTE)
516                 right.value--;

518         if (unop->type == EXPR_PREOP) {
519                 right.value++;
520                 estate = alloc_estate_range(right, start);
521                 if (estate_has_hard_max(sm->state))
522                         estate_set_hard_max(estate);
```

```
523                 estate_copy_fuzzy_max(estate, sm->state);
524                 set_extra_expr_mod(iter_var, estate);
525         }
526         if (unop->type == EXPR_POSTOP) {
527                 estate = alloc_estate_range(right, start);
528                 if (estate_has_hard_max(sm->state))
529                         estate_set_hard_max(estate);
530                 estate_copy_fuzzy_max(estate, sm->state);
531                 set_extra_expr_mod(iter_var, estate);
532         }
533         return get_sm_state_expr(SMATCH_EXTRA, iter_var);
534 }

536 static struct sm_state *handle_canonical_for_inc(struct expression *iter_expr,
537                                                  struct expression *condition)
538 {
539         struct expression *iter_var;
540         struct sm_state *sm;
541         struct smatch_state *estate;
542         sval_t start, end, max;

544         iter_var = iter_expr->unop;
545         sm = get_sm_state_expr(SMATCH_EXTRA, iter_var);
546         if (!sm)
547                 return NULL;
548         if (!estate_get_single_value(sm->state, &start))
549                 return NULL;
550         if (get_implied_max(condition->right, &end))
551                 end = sval_cast(get_type(iter_var), end);
552         else
553                 end = sval_type_max(get_type(iter_var));

555         if (get_sm_state_expr(SMATCH_EXTRA, condition->left) != sm)
556                 return NULL;

558         switch (condition->op) {
559         case SPECIAL_UNSIGNED_LT:
560         case SPECIAL_NOTEQUAL:
561         case '<':
562                 if (!sval_is_min(end))
563                         end.value--;
564                 break;
565         case SPECIAL_UNSIGNED_LTE:
566         case SPECIAL_LTE:
567                 break;
568         default:
569                 return NULL;
570         }
571         if (sval_cmp(end, start) < 0)
572                 return NULL;
573         estate = alloc_estate_range(start, end);
574         if (get_hard_max(condition->right, &max)) {
575                 estate_set_hard_max(estate);
576                 if (condition->op == '<' ||
577                     condition->op == SPECIAL_UNSIGNED_LT ||
578                     condition->op == SPECIAL_NOTEQUAL)
579                         max.value--;
580                 estate_set_fuzzy_max(estate, max);
581         }
582         set_extra_expr_mod(iter_var, estate);
583         return get_sm_state_expr(SMATCH_EXTRA, iter_var);
584 }

586 static struct sm_state *handle_canonical_for_dec(struct expression *iter_expr,
587                                                  struct expression *condition)
588 {
```



```
589         struct expression *iter_var;
590         struct sm_state *sm;
591         struct smatch_state *estate;
592         sval_t start, end;

594         iter_var = iter_expr->unop;
595         sm = get_sm_state_expr(SMATCH_EXTRA, iter_var);
596         if (!sm)
597                 return NULL;
598         if (!estate_get_single_value(sm->state, &start))
599                 return NULL;
600         if (!get_implied_min(condition->right, &end))
601                 end = sval_type_min(get_type(iter_var));
602         if (get_sm_state_expr(SMATCH_EXTRA, condition->left) != sm)
603                 return NULL;

605         switch (condition->op) {
606         case SPECIAL_NOTEQUAL:
607         case '>':
608                 if (!sval_is_min(end) && !sval_is_max(end))
609                         end.value++;
610                 break;
611         case SPECIAL_GTE:
612                 break;
613         default:
614                 return NULL;
615         }
616         if (sval_cmp(end, start) > 0)
617                 return NULL;
618         estate = alloc_estate_range(end, start);
619         estate_set_hard_max(estate);
620         estate_set_fuzzy_max(estate, estate_get_fuzzy_max(estate));
621         set_extra_expr_mod(iter_var, estate);
622         return get_sm_state_expr(SMATCH_EXTRA, iter_var);
623 }

625 static struct sm_state *handle_canonical_for_loops(struct statement *loop)
626 {
627         struct expression *iter_expr;
628         struct expression *condition;

630         if (!loop->iterator_post_statement)
631                 return NULL;
632         if (loop->iterator_post_statement->type != STMT_EXPRESSION)
633                 return NULL;
634         iter_expr = loop->iterator_post_statement->expression;
635         if (!loop->iterator_pre_condition)
636                 return NULL;
637         if (loop->iterator_pre_condition->type != EXPR_COMPARE)
638                 return NULL;
639         condition = loop->iterator_pre_condition;

641         if (iter_expr->op == SPECIAL_INCREMENT)
642                 return handle_canonical_for_inc(iter_expr, condition);
643         if (iter_expr->op == SPECIAL_DECREMENT)
644                 return handle_canonical_for_dec(iter_expr, condition);
645         return NULL;
646 }

648 struct sm_state *__extra_handle_canonical_loops(struct statement *loop, struct s
649 {
650         struct sm_state *ret;

652         /*
653          * Canonical loops are a hack.  The proper way to handle this is to
654          * use two passes, but unfortunately, doing two passes makes parsing
```

```
655          * code twice as slow.
656          *
657          * What we do is we set the inside state here, which overwrites whatever
658          * __extra_match_condition() does.  Then we set the outside state in
659          * __extra_pre_loop_hook_after().
660          *
661          */
662         __push_fake_cur_stree();
663         if (!loop->iterator_post_statement)
664                 ret = handle_canonical_while_count_down(loop);
665         else
666                 ret = handle_canonical_for_loops(loop);
667         *stree = __pop_fake_cur_stree();
668         return ret;
669 }

671 int __iterator_unchanged(struct sm_state *sm)
672 {
673         if (!sm)
674                 return 0;
675         if (get_sm_state(my_id, sm->name, sm->sym) == sm)
676                 return 1;
677         return 0;
678 }

680 static void while_count_down_after(struct sm_state *sm, struct expression *condi
681 {
682         struct expression *unop;
683         int op;
684         sval_t limit, after_value;

686         if (!get_countdown_info(condition, &unop, &op, &limit))
687                 return;
688         after_value = estate_min(sm->state);
689         after_value.value--;
690         set_extra_mod(sm->name, sm->sym, condition->unop, alloc_estate_sval(afte
691 }

693 void __extra_pre_loop_hook_after(struct sm_state *sm,
694                                  struct statement *iterator,
695                                  struct expression *condition)
696 {
697         struct expression *iter_expr;
698         sval_t limit;
699         struct smatch_state *state;

701         if (!iterator) {
702                 while_count_down_after(sm, condition);
703                 return;
704         }

706         iter_expr = iterator->expression;

708         if (condition->type != EXPR_COMPARE)
709                 return;
710         if (iter_expr->op == SPECIAL_INCREMENT) {
711                 limit = sval_binop(estate_max(sm->state), '+',
712                                    sval_type_val(estate_type(sm->state), 1));
713         } else {
714                 limit = sval_binop(estate_min(sm->state), '-',
715                                    sval_type_val(estate_type(sm->state), 1));
716         }
717         if (!estate_has_hard_max(sm->state) && !__has_breaks()) {
718                 if (iter_expr->op == SPECIAL_INCREMENT)
719                         state = alloc_estate_range(estate_min(sm->state), limit)
720                 else
```



```
721                         state = alloc_estate_range(limit, estate_max(sm->state))
722         } else {
723                 state = alloc_estate_sval(limit);
724         }
725         if (!estate_has_hard_max(sm->state)) {
726                 estate_clear_hard_max(state);
727         }
728         if (estate_has_fuzzy_max(sm->state)) {
729                 sval_t hmax = estate_get_fuzzy_max(sm->state);
730                 sval_t max = estate_max(sm->state);

732                 if (sval_cmp(hmax, max) != 0)
733                         estate_clear_fuzzy_max(state);
734         } else if (!estate_has_fuzzy_max(sm->state)) {
735                 estate_clear_fuzzy_max(state);
736         }

738         set_extra_mod(sm->name, sm->sym, iter_expr, state);
739 }

741 static struct stree *unmatched_stree;
742 static struct smatch_state *unmatched_state(struct sm_state *sm)
743 {
744         struct smatch_state *state;

746         if (unmatched_stree) {
747                 state = get_state_stree(unmatched_stree, SMATCH_EXTRA, sm->name,
748                 if (state)
749                         return state;
750         }
751         if (parent_is_gone_var_sym(sm->name, sm->sym))
752                 return alloc_estate_empty();
753         return alloc_estate_whole(estate_type(sm->state));
754 }

756 static void clear_the_pointed_at(struct expression *expr)
757 {
758         struct stree *stree;
759         char *name;
760         struct symbol *sym;
761         struct sm_state *tmp;

763         name = expr_to_var_sym(expr, &sym);
764         if (!name || !sym)
765                 goto free;

767         stree = __get_cur_stree();
768         FOR_EACH_MY_SM(SMATCH_EXTRA, stree, tmp) {
769                 if (tmp->name[0] != '*')
770                         continue;
771                 if (tmp->sym != sym)
772                         continue;
773                 if (strcmp(tmp->name + 1, name) != 0)
774                         continue;
775                 set_extra_mod(tmp->name, tmp->sym, expr, alloc_estate_whole(esta
776         } END_FOR_EACH_SM(tmp);

778 free:
779         free_string(name);
780 }

782 static int is_const_param(struct expression *expr, int param)
783 {
784         struct symbol *type;

786         type = get_arg_type(expr, param);
```

```
787         if (!type)
788                 return 0;
789         if (type->ctype.modifiers & MOD_CONST)
790                 return 1;
791         return 0;
792 }

794 static void match_function_call(struct expression *expr)
795 {
796         struct expression *arg;
797         struct expression *tmp;
798         int param = -1;

800         /* if we have the db this is handled in smatch_function_hooks.c */
801         if (!option_no_db)
802                 return;
803         if (inlinable(expr->fn))
804                 return;

806         FOR_EACH_PTR(expr->args, arg) {
807                 param++;
808                 if (is_const_param(expr->fn, param))
809                         continue;
810                 tmp = strip_expr(arg);
811                 if (tmp->type == EXPR_PREOP && tmp->op == '&')
812                         set_extra_expr_mod(tmp->unop, alloc_estate_whole(get_typ
813                 else
814                         clear_the_pointed_at(tmp);
815         } END_FOR_EACH_PTR(arg);
816 }

818 static int values_fit_type(struct expression *left, struct expression *right)
819 {
820         struct range_list *rl;
821         struct symbol *type;

823         type = get_type(left);
824         if (!type)
825                 return 0;
826         get_absolute_rl(right, &rl);
827         if (type_unsigned(type) && sval_is_negative(rl_min(rl)))
828                 return 0;
829         if (sval_cmp(sval_type_min(type), rl_min(rl)) > 0)
830                 return 0;
831         if (sval_cmp(sval_type_max(type), rl_max(rl)) < 0)
832                 return 0;
833         return 1;
834 }

836 static void save_chunk_info(struct expression *left, struct expression *right)
837 {
838         struct var_sym_list *vsl;
839         struct var_sym *vs;
840         struct expression *add_expr;
841         struct symbol *type;
842         sval_t sval;
843         char *name;
844         struct symbol *sym;

846         if (right->type != EXPR_BINOP || right->op != '-')
847                 return;
848         if (!get_value(right->left, &sval))
849                 return;
850         if (!expr_to_sym(right->right))
851                 return;
```



```
853         add_expr = binop_expression(left, '+', right->right);
854         type = get_type(add_expr);
855         if (!type)
856                 return;
857         name = expr_to_chunk_sym_vsl(add_expr, &sym, &vsl);
858         if (!name || !vsl)
859                 goto free;
860         FOR_EACH_PTR(vsl, vs) {
861                 store_link(link_id, vs->var, vs->sym, name, sym);
862         } END_FOR_EACH_PTR(vs);

864         set_state(SMATCH_EXTRA, name, sym, alloc_estate_sval(sval_cast(type, sva
865 free:
866         free_string(name);
867 }

869 static void do_array_assign(struct expression *left, int op, struct expression *
870 {
871         struct range_list *rl;

873         if (op == '=') {
874                 get_absolute_rl(right, &rl);
875                 rl = cast_rl(get_type(left), rl);
876         } else {
877                 rl = alloc_whole_rl(get_type(left));
878         }

880         set_extra_array_mod(left, alloc_estate_rl(rl));
881 }

883 static void match_vanilla_assign(struct expression *left, struct expression *rig
884 {
885         struct range_list *orig_rl = NULL;
886         struct range_list *rl = NULL;
887         struct symbol *right_sym;
888         struct symbol *left_type;
889         struct symbol *right_type;
890         char *right_name = NULL;
891         struct symbol *sym;
892         char *name;
893         sval_t sval, max;
894         struct smatch_state *state;
895         int comparison;

897         if (is_struct(left))
898                 return;

900         save_chunk_info(left, right);

902         name = expr_to_var_sym(left, &sym);
903         if (!name) {
904                 if (chunk_has_array(left))
905                         do_array_assign(left, '=', right);
906                 return;
907         }

909         left_type = get_type(left);
910         right_type = get_type(right);

912         right_name = expr_to_var_sym(right, &right_sym);

914         if (!__in_fake_assign &&
915             !(right->type == EXPR_PREOP && right->op == '&') &&
916             right_name && right_sym &&
917             values_fit_type(left, strip_expr(right)) &&
918             !has_symbol(right, sym)) {
```

```
919                 set_equiv(left, right);
920                 goto free;
921         }

923         if (is_pointer(right) && get_address_rl(right, &rl)) {
924                 state = alloc_estate_rl(rl);
925                 goto done;
926         }

928         if (get_implied_value(right, &sval)) {
929                 state = alloc_estate_sval(sval_cast(left_type, sval));
930                 goto done;
931         }

933         if (__in_fake_assign) {
934                 struct smatch_state *right_state;
935                 sval_t sval;

937                 if (get_value(right, &sval)) {
938                         sval = sval_cast(left_type, sval);
939                         state = alloc_estate_sval(sval);
940                         goto done;
941                 }

943                 right_state = get_state(SMATCH_EXTRA, right_name, right_sym);
944                 if (right_state) {
945                         /* simple assignment */
946                         state = clone_estate(right_state);
947                         goto done;
948                 }

950                 state = alloc_estate_rl(alloc_whole_rl(left_type));
951                 goto done;
952         }

954         comparison = get_comparison(left, right);
955         if (comparison) {
956                 comparison = flip_comparison(comparison);
957                 get_implied_rl(left, &orig_rl);
958         }

960         if (get_implied_rl(right, &rl)) {
961                 rl = cast_rl(left_type, rl);
962                 if (orig_rl)
963                         filter_by_comparison(&rl, comparison, orig_rl);
964                 state = alloc_estate_rl(rl);
965                 if (get_hard_max(right, &max)) {
966                         estate_set_hard_max(state);
967                         estate_set_fuzzy_max(state, max);
968                 }
969         } else {
970                 rl = alloc_whole_rl(right_type);
971                 rl = cast_rl(left_type, rl);
972                 if (orig_rl)
973                         filter_by_comparison(&rl, comparison, orig_rl);
974                 state = alloc_estate_rl(rl);
975         }

977 done:
978         set_extra_mod(name, sym, left, state);
979 free:
980         free_string(right_name);
981 }

983 static int op_remove_assign(int op)
984 {
```



```
985         switch (op) {
986         case SPECIAL_ADD_ASSIGN:
987                 return '+';
988         case SPECIAL_SUB_ASSIGN:
989                 return '-';
990         case SPECIAL_MUL_ASSIGN:
991                 return '*';
992         case SPECIAL_DIV_ASSIGN:
993                 return '/';
994         case SPECIAL_MOD_ASSIGN:
995                 return '%';
996         case SPECIAL_AND_ASSIGN:
997                 return '&';
998         case SPECIAL_OR_ASSIGN:
999                 return '|';
1000        case SPECIAL_XOR_ASSIGN:
1001                return '^';
1002        case SPECIAL_SHL_ASSIGN:
1003                return SPECIAL_LEFTSHIFT;
1004        case SPECIAL_SHR_ASSIGN:
1005                return SPECIAL_RIGHTSHIFT;
1006        default:
1007                return op;
1008        }
1009 }

1011 static void match_assign(struct expression *expr)
1012 {
1013        struct range_list *rl = NULL;
1014        struct expression *left;
1015        struct expression *right;
1016        struct expression *binop_expr;
1017        struct symbol *left_type;
1018        struct symbol *sym;
1019        char *name;
1020        sval_t left_min, left_max;
1021        sval_t right_min, right_max;
1022        sval_t res_min, res_max;

1024        left = strip_expr(expr->left);

1026        right = strip_parens(expr->right);
1027        if (right->type == EXPR_CALL && sym_name_is("__builtin_expect", right->f
1028                right = get_argument_from_call_expr(right->args, 0);
1029        while (right->type == EXPR_ASSIGNMENT && right->op == '=')
1030                right = strip_parens(right->left);

1032        if (expr->op == '=' && is_condition(expr->right))
1033                return; /* handled in smatch_condition.c */
1034        if (expr->op == '=' && right->type == EXPR_CALL)
1035                return; /* handled in smatch_function_hooks.c */
1036        if (expr->op == '=') {
1037                match_vanilla_assign(left, right);
1038                return;
1039        }

1041        name = expr_to_var_sym(left, &sym);
1042        if (!name)
1043                return;

1045        left_type = get_type(left);

1047        res_min = sval_type_min(left_type);
1048        res_max = sval_type_max(left_type);

1050        switch (expr->op) {
```

```
1051         case SPECIAL_ADD_ASSIGN:
1052                 get_absolute_max(left, &left_max);
1053                 get_absolute_max(right, &right_max);
1054                 if (sval_binop_overflows(left_max, '+', sval_cast(left_type, rig
1055                         break;
1056                 if (get_implied_min(left, &left_min) &&
1057                     !sval_is_negative_min(left_min) &&
1058                     get_implied_min(right, &right_min) &&
1059                     !sval_is_negative_min(right_min)) {
1060                         res_min = sval_binop(left_min, '+', right_min);
1061                         res_min = sval_cast(left_type, res_min);
1062                 }
1063                 if (inside_loop())  /* we are assuming loops don't lead to wrapp
1064                         break;
1065                 res_max = sval_binop(left_max, '+', right_max);
1066                 res_max = sval_cast(left_type, res_max);
1067                 break;
1068         case SPECIAL_SUB_ASSIGN:
1069                 if (get_implied_max(left, &left_max) &&
1070                     !sval_is_max(left_max) &&
1071                     get_implied_min(right, &right_min) &&
1072                     !sval_is_min(right_min)) {
1073                         res_max = sval_binop(left_max, '-', right_min);
1074                         res_max = sval_cast(left_type, res_max);
1075                 }
1076                 if (inside_loop())
1077                         break;
1078                 if (get_implied_min(left, &left_min) &&
1079                     !sval_is_min(left_min) &&
1080                     get_implied_max(right, &right_max) &&
1081                     !sval_is_max(right_max)) {
1082                         res_min = sval_binop(left_min, '-', right_max);
1083                         res_min = sval_cast(left_type, res_min);
1084                 }
1085                 break;
1086         case SPECIAL_AND_ASSIGN:
1087         case SPECIAL_MOD_ASSIGN:
1088         case SPECIAL_SHL_ASSIGN:
1089         case SPECIAL_SHR_ASSIGN:
1090         case SPECIAL_OR_ASSIGN:
1091         case SPECIAL_XOR_ASSIGN:
1092         case SPECIAL_MUL_ASSIGN:
1093         case SPECIAL_DIV_ASSIGN:
1094                 binop_expr = binop_expression(expr->left,
1095                                               op_remove_assign(expr->op),
1096                                               expr->right);
1097                 if (get_absolute_rl(binop_expr, &rl)) {
1098                         rl = cast_rl(left_type, rl);
1099                         set_extra_mod(name, sym, left, alloc_estate_rl(rl));
1100                         goto free;
1101                 }
1102                 break;
1103         }
1104         rl = cast_rl(left_type, alloc_rl(res_min, res_max));
1105         set_extra_mod(name, sym, left, alloc_estate_rl(rl));
1106 free:
1107         free_string(name);
1108 }

1110 static struct smatch_state *increment_state(struct smatch_state *state)
1111 {
1112         sval_t min = estate_min(state);
1113         sval_t max = estate_max(state);

1115         if (!estate_rl(state))
1116                 return NULL;
```



```
1118         if (inside_loop())
1119                 max = sval_type_max(max.type);

1121         if (!sval_is_min(min) && !sval_is_max(min))
1122                 min.value++;
1123         if (!sval_is_min(max) && !sval_is_max(max))
1124                 max.value++;
1125         return alloc_estate_range(min, max);
1126 }

1128 static struct smatch_state *decrement_state(struct smatch_state *state)
1129 {
1130         sval_t min = estate_min(state);
1131         sval_t max = estate_max(state);

1133         if (!estate_rl(state))
1134                 return NULL;

1136         if (inside_loop())
1137                 min = sval_type_min(min.type);

1139         if (!sval_is_min(min) && !sval_is_max(min))
1140                 min.value--;
1141         if (!sval_is_min(max) && !sval_is_max(max))
1142                 max.value--;
1143         return alloc_estate_range(min, max);
1144 }

1146 static void clear_pointed_at_state(struct expression *expr)
1147 {
1148         struct symbol *type;

1150         /*
1151          * ALERT: This is sort of a mess.  If it's is a struct assigment like
1152          * "foo = bar;", then that's handled by smatch_struct_assignment.c.
1153          * the same thing for p++ where "p" is a struct.  Most modifications
1154          * are handled by the assignment hook or the db.  Smatch_extra.c doesn't
1155          * use smatch_modification.c because we have to get the ordering right
1156          * or something.  So if you have p++ where p is a pointer to a standard
1157          * c type then we handle that here.  What a mess.
1158          */
1159         expr = strip_expr(expr);
1160         type = get_type(expr);
1161         if (!type || type->type != SYM_PTR)
1162                 return;
1163         type = get_real_base_type(type);
1164         if (!type || type->type != SYM_BASETYPE)
1165                 return;
1166         set_extra_expr_nomod(deref_expression(expr), alloc_estate_whole(type));
1167 }

1169 static void unop_expr(struct expression *expr)
1170 {
1171         struct smatch_state *state;

1173         if (expr->smatch_flags & Handled)
1174                 return;

1176         switch (expr->op) {
1177         case SPECIAL_INCREMENT:
1178                 state = get_state_expr(SMATCH_EXTRA, expr->unop);
1179                 state = increment_state(state);
1180                 if (!state)
1181                         state = alloc_estate_whole(get_type(expr));
1182                 set_extra_expr_mod(expr->unop, state);
```

```
1183                 clear_pointed_at_state(expr->unop);
1184                 break;
1185         case SPECIAL_DECREMENT:
1186                 state = get_state_expr(SMATCH_EXTRA, expr->unop);
1187                 state = decrement_state(state);
1188                 if (!state)
1189                         state = alloc_estate_whole(get_type(expr));
1190                 set_extra_expr_mod(expr->unop, state);
1191                 clear_pointed_at_state(expr->unop);
1192                 break;
1193         default:
1194                 return;
1195         }
1196 }

1198 static void asm_expr(struct statement *stmt)
1199 {

1201         struct expression *expr;
1202         struct symbol *type;
1203         int state = 0;

1205         FOR_EACH_PTR(stmt->asm_outputs, expr) {
1206                 switch (state) {
1207                 case 0: /* identifier */
1208                 case 1: /* constraint */
1209                         state++;
1210                         continue;
1211                 case 2: /* expression */
1212                         state = 0;
1213                         type = get_type(strip_expr(expr));
1214                         set_extra_expr_mod(expr, alloc_estate_whole(type));
1215                         continue;
1216                 }
1217         } END_FOR_EACH_PTR(expr);
1218 }

1220 static void check_dereference(struct expression *expr)
1221 {
1222         struct smatch_state *state;

1224         if (__in_fake_assign)
1225                 return;
1226         if (outside_of_function())
1227                 return;
1228         state = get_extra_state(expr);
1229         if (state) {
1230                 struct range_list *rl;

1232                 rl = rl_intersection(estate_rl(state), valid_ptr_rl);
1233                 if (rl_equiv(rl, estate_rl(state)))
1234                         return;
1235                 set_extra_expr_nomod(expr, alloc_estate_rl(rl));
1236         } else {
1237                 set_extra_expr_nomod(expr, alloc_estate_range(valid_ptr_min_sval
1238         }
1239 }

1241 static void match_dereferences(struct expression *expr)
1242 {
1243         if (expr->type != EXPR_PREOP)
1244                 return;
1245         /* it's saying that foo[1] = bar dereferences foo[1] */
1246         if (is_array(expr))
1247                 return;
1248         check_dereference(expr->unop);
```



```
1249 }

1251 static void match_pointer_as_array(struct expression *expr)
1252 {
1253         if (!is_array(expr))
1254                 return;
1255         check_dereference(get_array_base(expr));
1256 }

1258 static void find_dereferences(struct expression *expr)
1259 {
1260         while (expr->type == EXPR_PREOP) {
1261                 if (expr->op == '*')
1262                         check_dereference(expr->unop);
1263                 expr = strip_expr(expr->unop);
1264         }
1265 }

1267 static void set_param_dereferenced(struct expression *call, struct expression *a
1268 {
1269         struct symbol *sym;
1270         char *name;

1272         name = get_variable_from_key(arg, key, &sym);
1273         if (name && sym) {
1274                 struct smatch_state *orig, *new;
1275                 struct range_list *rl;

1277                 orig = get_state(SMATCH_EXTRA, name, sym);
1278                 if (orig) {
1279                         rl = rl_intersection(estate_rl(orig),
1280                                              alloc_rl(valid_ptr_min_sval,
1281                                                       valid_ptr_max_sval));
1282                         new = alloc_estate_rl(rl);
1283                 } else {
1284                         new = alloc_estate_range(valid_ptr_min_sval, valid_ptr_m
1285                 }

1287                 set_extra_nomod(name, sym, NULL, new);
1288         }
1289         free_string(name);

1291         find_dereferences(arg);
1292 }

1294 static sval_t add_one(sval_t sval)
1295 {
1296         sval.value++;
1297         return sval;
1298 }

1300 static int handle_postop_inc(struct expression *left, int op, struct expression
1301 {
1302         struct statement *stmt;
1303         struct expression *cond;
1304         struct smatch_state *true_state, *false_state;
1305         sval_t start;
1306         sval_t limit;

1308         /*
1309          * If we're decrementing here then that's a canonical while count down
1310          * so it's handled already.  We're only handling loops like:
1311          * i = 0;
1312          * do { ... } while (i++ < 3);
1313          */
```

```
1315         if (left->type != EXPR_POSTOP || left->op != SPECIAL_INCREMENT)
1316                 return 0;

1318         stmt = __cur_stmt->parent;
1319         if (!stmt)
1320                 return 0;
1321         if (stmt->type == STMT_COMPOUND)
1322                 stmt = stmt->parent;
1323         if (!stmt || stmt->type != STMT_ITERATOR || !stmt->iterator_post_conditi
1324                 return 0;

1326         cond = strip_expr(stmt->iterator_post_condition);
1327         if (cond->type != EXPR_COMPARE || cond->op != op)
1328                 return 0;
1329         if (left != strip_expr(cond->left) || right != strip_expr(cond->right))
1330                 return 0;

1332         if (!get_implied_value(left->unop, &start))
1333                 return 0;
1334         if (!get_implied_value(right, &limit))
1335                 return 0;

1337         if (sval_cmp(start, limit) > 0)
1338                 return 0;

1340         switch (op) {
1341         case '<':
1342         case SPECIAL_UNSIGNED_LT:
1343                 break;
1344         case SPECIAL_LTE:
1345         case SPECIAL_UNSIGNED_LTE:
1346                 limit = add_one(limit);
1347         default:
1348                 return 0;

1350         }

1352         true_state = alloc_estate_range(add_one(start), limit);
1353         false_state = alloc_estate_range(add_one(limit), add_one(limit));

1355         /* Currently we just discard the false state but when two passes is
1356          * implimented correctly then it will use it.
1357          */

1359         set_extra_expr_true_false(left->unop, true_state, false_state);

1361         return 1;
1362 }

1364 bool is_impossible_variable(struct expression *expr)
1365 {
1366         struct smatch_state *state;

1368         state = get_extra_state(expr);
1369         if (state && !estate_rl(state))
1370                 return true;
1371         return false;
1372 }

1374 static void handle_comparison(struct symbol *type, struct expression *left, int
1375 {
1376         struct range_list *left_orig;
1377         struct range_list *left_true;
1378         struct range_list *left_false;
1379         struct range_list *right_orig;
1380         struct range_list *right_true;
```



```
1381         struct range_list *right_false;
1382         struct smatch_state *left_true_state;
1383         struct smatch_state *left_false_state;
1384         struct smatch_state *right_true_state;
1385         struct smatch_state *right_false_state;
1386         sval_t dummy, hard_max;
1387         int left_postop = 0;
1388         int right_postop = 0;

1390         if (left->op == SPECIAL_INCREMENT || left->op == SPECIAL_DECREMENT) {
1391                 if (left->type == EXPR_POSTOP) {
1392                         left->smatch_flags |= Handled;
1393                         left_postop = left->op;
1394                         if (handle_postop_inc(left, op, right))
1395                                 return;
1396                 }
1397                 left = strip_parens(left->unop);
1398         }
1399         while (left->type == EXPR_ASSIGNMENT)
1400                 left = strip_parens(left->left);

1402         if (right->op == SPECIAL_INCREMENT || right->op == SPECIAL_DECREMENT) {
1403                 if (right->type == EXPR_POSTOP) {
1404                         right->smatch_flags |= Handled;
1405                         right_postop = right->op;
1406                 }
1407                 right = strip_parens(right->unop);
1408         }

1410         if (is_impossible_variable(left) || is_impossible_variable(right))
1411                 return;

1413         get_real_absolute_rl(left, &left_orig);
1414         left_orig = cast_rl(type, left_orig);

1416         get_real_absolute_rl(right, &right_orig);
1417         right_orig = cast_rl(type, right_orig);

1419         split_comparison_rl(left_orig, op, right_orig, &left_true, &left_false,

1421         left_true = rl_truncate_cast(get_type(strip_expr(left)), left_true);
1422         left_false = rl_truncate_cast(get_type(strip_expr(left)), left_false);
1423         right_true = rl_truncate_cast(get_type(strip_expr(right)), right_true);
1424         right_false = rl_truncate_cast(get_type(strip_expr(right)), right_false)

1426         if (!left_true || !left_false) {
1427                 struct range_list *tmp_true, *tmp_false;

1429                 split_comparison_rl(alloc_whole_rl(type), op, right_orig, &tmp_t
1430                 tmp_true = rl_truncate_cast(get_type(strip_expr(left)), tmp_true
1431                 tmp_false = rl_truncate_cast(get_type(strip_expr(left)), tmp_fal
1432                 if (tmp_true && tmp_false)
1433                         __save_imaginary_state(left, tmp_true, tmp_false);
1434         }

1436         if (!right_true || !right_false) {
1437                 struct range_list *tmp_true, *tmp_false;

1439                 split_comparison_rl(alloc_whole_rl(type), op, right_orig, NULL,
1440                 tmp_true = rl_truncate_cast(get_type(strip_expr(right)), tmp_tru
1441                 tmp_false = rl_truncate_cast(get_type(strip_expr(right)), tmp_fa
1442                 if (tmp_true && tmp_false)
1443                         __save_imaginary_state(right, tmp_true, tmp_false);
1444         }

1446         left_true_state = alloc_estate_rl(left_true);
```

```
1447         left_false_state = alloc_estate_rl(left_false);
1448         right_true_state = alloc_estate_rl(right_true);
1449         right_false_state = alloc_estate_rl(right_false);
1450
1451         switch (op) {
1452         case '<':
1453         case SPECIAL_UNSIGNED_LT:
1454         case SPECIAL_UNSIGNED_LTE:
1455         case SPECIAL_LTE:
1456                 if (get_hard_max(right, &dummy))
1457                         estate_set_hard_max(left_true_state);
1458                 if (get_hard_max(left, &dummy))
1459                         estate_set_hard_max(right_false_state);
1460                 break;
1461         case '>':
1462         case SPECIAL_UNSIGNED_GT:
1463         case SPECIAL_UNSIGNED_GTE:
1464         case SPECIAL_GTE:
1465                 if (get_hard_max(left, &dummy))
1466                         estate_set_hard_max(right_true_state);
1467                 if (get_hard_max(right, &dummy))
1468                         estate_set_hard_max(left_false_state);
1469                 break;
1470         }
1471
1472         switch (op) {
1473         case '<':
1474         case SPECIAL_UNSIGNED_LT:
1475         case SPECIAL_UNSIGNED_LTE:
1476         case SPECIAL_LTE:
1477                 if (get_hard_max(right, &hard_max)) {
1478                         if (op == '<' || op == SPECIAL_UNSIGNED_LT)
1479                                 hard_max.value--;
1480                         estate_set_fuzzy_max(left_true_state, hard_max);
1481                 }
1482                 if (get_implied_value(right, &hard_max)) {
1483                         if (op == SPECIAL_UNSIGNED_LTE ||
1484                             op == SPECIAL_LTE)
1485                                 hard_max.value++;
1486                         estate_set_fuzzy_max(left_false_state, hard_max);
1487                 }
1488                 if (get_hard_max(left, &hard_max)) {
1489                         if (op == SPECIAL_UNSIGNED_LTE ||
1490                             op == SPECIAL_LTE)
1491                                 hard_max.value--;
1492                         estate_set_fuzzy_max(right_false_state, hard_max);
1493                 }
1494                 if (get_implied_value(left, &hard_max)) {
1495                         if (op == '<' || op == SPECIAL_UNSIGNED_LT)
1496                                 hard_max.value++;
1497                         estate_set_fuzzy_max(right_true_state, hard_max);
1498                 }
1499                 break;
1500         case '>':
1501         case SPECIAL_UNSIGNED_GT:
1502         case SPECIAL_UNSIGNED_GTE:
1503         case SPECIAL_GTE:
1504                 if (get_hard_max(left, &hard_max)) {
1505                         if (op == '>' || op == SPECIAL_UNSIGNED_GT)
1506                                 hard_max.value--;
1507                         estate_set_fuzzy_max(right_true_state, hard_max);
1508                 }
1509                 if (get_implied_value(left, &hard_max)) {
1510                         if (op == SPECIAL_UNSIGNED_GTE ||
1511                             op == SPECIAL_GTE)
1512                                 hard_max.value++;
```



```
1513                         estate_set_fuzzy_max(right_false_state, hard_max);
1514                 }
1515                 if (get_hard_max(right, &hard_max)) {
1516                         if (op == SPECIAL_UNSIGNED_LTE ||
1517                             op == SPECIAL_LTE)
1518                                 hard_max.value--;
1519                         estate_set_fuzzy_max(left_false_state, hard_max);
1520                 }
1521                 if (get_implied_value(right, &hard_max)) {
1522                         if (op == '>' ||
1523                             op == SPECIAL_UNSIGNED_GT)
1524                                 hard_max.value++;
1525                         estate_set_fuzzy_max(left_true_state, hard_max);
1526                 }
1527                 break;
1528         case SPECIAL_EQUAL:
1529                 if (get_hard_max(left, &hard_max))
1530                         estate_set_fuzzy_max(right_true_state, hard_max);
1531                 if (get_hard_max(right, &hard_max))
1532                         estate_set_fuzzy_max(left_true_state, hard_max);
1533                 break;
1534         }
1535
1536         if (get_hard_max(left, &hard_max)) {
1537                 estate_set_hard_max(left_true_state);
1538                 estate_set_hard_max(left_false_state);
1539         }
1540         if (get_hard_max(right, &hard_max)) {
1541                 estate_set_hard_max(right_true_state);
1542                 estate_set_hard_max(right_false_state);
1543         }
1544
1545         if (left_postop == SPECIAL_INCREMENT) {
1546                 left_true_state = increment_state(left_true_state);
1547                 left_false_state = increment_state(left_false_state);
1548         }
1549         if (left_postop == SPECIAL_DECREMENT) {
1550                 left_true_state = decrement_state(left_true_state);
1551                 left_false_state = decrement_state(left_false_state);
1552         }
1553         if (right_postop == SPECIAL_INCREMENT) {
1554                 right_true_state = increment_state(right_true_state);
1555                 right_false_state = increment_state(right_false_state);
1556         }
1557         if (right_postop == SPECIAL_DECREMENT) {
1558                 right_true_state = decrement_state(right_true_state);
1559                 right_false_state = decrement_state(right_false_state);
1560         }
1561
1562         if (estate_rl(left_true_state) && estates_equiv(left_true_state, left_fa
1563                 left_true_state = NULL;
1564                 left_false_state = NULL;
1565         }
1566
1567         if (estate_rl(right_true_state) && estates_equiv(right_true_state, right
1568                 right_true_state = NULL;
1569                 right_false_state = NULL;
1570         }
1571
1572         /* Don't introduce new states for known true/false conditions */
1573         if (rl_equiv(left_orig, estate_rl(left_true_state)))
1574                 left_true_state = NULL;
1575         if (rl_equiv(left_orig, estate_rl(left_false_state)))
1576                 left_false_state = NULL;
1577         if (rl_equiv(right_orig, estate_rl(right_true_state)))
1578                 right_true_state = NULL;
```

```
1579         if (rl_equiv(right_orig, estate_rl(right_false_state)))
1580                 right_false_state = NULL;

1582         set_extra_expr_true_false(left, left_true_state, left_false_state);
1583         set_extra_expr_true_false(right, right_true_state, right_false_state);
1584 }

1586 static int is_simple_math(struct expression *expr)
1587 {
1588         if (!expr)
1589                 return 0;
1590         if (expr->type != EXPR_BINOP)
1591                 return 0;
1592         switch (expr->op) {
1593         case '+':
1594         case '-':
1595         case '*':
1596                 return 1;
1597         }
1598         return 0;
1599 }

1601 static void move_known_values(struct expression **left_p, struct expression **ri
1602 {
1603         struct expression *left = *left_p;
1604         struct expression *right = *right_p;
1605         sval_t sval, dummy;

1607         if (get_implied_value(left, &sval)) {
1608                 if (!is_simple_math(right))
1609                         return;
1610                 if (get_implied_value(right, &dummy))
1611                         return;
1612                 if (right->op == '*') {
1613                         sval_t divisor;

1615                         if (!get_value(right->right, &divisor))
1616                                 return;
1617                         if (divisor.value == 0)
1618                                 return;
1619                         *left_p = binop_expression(left, invert_op(right->op), r
1620                         *right_p = right->left;
1621                         return;
1622                 }
1623                 if (right->op == '+' && get_value(right->left, &sval)) {
1624                         *left_p = binop_expression(left, invert_op(right->op), r
1625                         *right_p = right->right;
1626                         return;
1627                 }
1628                 if (get_value(right->right, &sval)) {
1629                         *left_p = binop_expression(left, invert_op(right->op), r
1630                         *right_p = right->left;
1631                         return;
1632                 }
1633                 return;
1634         }
1635         if (get_implied_value(right, &sval)) {
1636                 if (!is_simple_math(left))
1637                         return;
1638                 if (get_implied_value(left, &dummy))
1639                         return;
1640                 if (left->op == '*') {
1641                         sval_t divisor;

1643                         if (!get_value(left->right, &divisor))
1644                                 return;
```



```
1645                         if (divisor.value == 0)
1646                                 return;
1647                         *right_p = binop_expression(right, invert_op(left->op),
1648                         *left_p = left->left;
1649                         return;
1650                 }
1651                 if (left->op == '+' && get_value(left->left, &sval)) {
1652                         *right_p = binop_expression(right, invert_op(left->op),
1653                         *left_p = left->right;
1654                         return;
1655                 }

1657                 if (get_value(left->right, &sval)) {
1658                         *right_p = binop_expression(right, invert_op(left->op),
1659                         *left_p = left->left;
1660                         return;
1661                 }
1662                 return;
1663         }
1664 }

1666 /*
1667  * The reason for do_simple_algebra() is to solve things like:
1668  * if (foo > 66 || foo + bar > 64) {
1669  * "foo" is not really a known variable so it won't be handled by
1670  * move_known_variables() but it's a super common idiom.
1671  *
1672  */
1673 static int do_simple_algebra(struct expression **left_p, struct expression **rig
1674 {
1675         struct expression *left = *left_p;
1676         struct expression *right = *right_p;
1677         struct range_list *rl;
1678         sval_t tmp;

1680         if (left->type != EXPR_BINOP || left->op != '+')
1681                 return 0;
1682         if (can_integer_overflow(get_type(left), left))
1683                 return 0;
1684         if (!get_implied_value(right, &tmp))
1685                 return 0;

1687         if (!get_implied_value(left->left, &tmp) &&
1688             get_implied_rl(left->left, &rl) &&
1689             !is_whole_rl(rl)) {
1690                 *right_p = binop_expression(right, '-', left->left);
1691                 *left_p = left->right;
1692                 return 1;
1693         }
1694         if (!get_implied_value(left->right, &tmp) &&
1695             get_implied_rl(left->right, &rl) &&
1696             !is_whole_rl(rl)) {
1697                 *right_p = binop_expression(right, '-', left->right);
1698                 *left_p = left->left;
1699                 return 1;
1700         }

1702         return 0;
1703 }

1705 static int match_func_comparison(struct expression *expr)
1706 {
1707         struct expression *left = strip_expr(expr->left);
1708         struct expression *right = strip_expr(expr->right);
1709         sval_t sval;
```

```
1711         /*
1712          * fixme: think about this harder. We should always be trying to limit
1713          * the non-call side as well.  If we can't determine the limitter does
1714          * that mean we aren't querying the database and are missing important
1715          * information?
1716          */
1717
1718         if (left->type == EXPR_CALL) {
1719                 if (get_implied_value(left, &sval)) {
1720                         handle_comparison(get_type(expr), left, expr->op, right)
1721                         return 1;
1722                 }
1723                 function_comparison(left, expr->op, right);
1724                 return 1;
1725         }
1726
1727         if (right->type == EXPR_CALL) {
1728                 if (get_implied_value(right, &sval)) {
1729                         handle_comparison(get_type(expr), left, expr->op, right)
1730                         return 1;
1731                 }
1732                 function_comparison(left, expr->op, right);
1733                 return 1;
1734         }
1735
1736         return 0;
1737 }
1738
1739 /* Handle conditions like "if (foo + bar < foo) {" */
1740 static int handle_integer_overflow_test(struct expression *expr)
1741 {
1742         struct expression *left, *right;
1743         struct symbol *type;
1744         sval_t left_min, right_min, min, max;
1745
1746         if (expr->op != '<' && expr->op != SPECIAL_UNSIGNED_LT)
1747                 return 0;
1748
1749         left = strip_parens(expr->left);
1750         right = strip_parens(expr->right);
1751
1752         if (left->op != '+')
1753                 return 0;
1754
1755         type = get_type(expr);
1756         if (!type)
1757                 return 0;
1758         if (type_positive_bits(type) == 32) {
1759                 max.type = &uint_ctype;
1760                 max.uvalue = (unsigned int)-1;
1761         } else if (type_positive_bits(type) == 64) {
1762                 max.type = &ulong_ctype;
1763                 max.value = (unsigned long long)-1;
1764         } else {
1765                 return 0;
1766         }
1767
1768         if (!expr_equiv(left->left, right) && !expr_equiv(left->right, right))
1769                 return 0;
1770
1771         get_absolute_min(left->left, &left_min);
1772         get_absolute_min(left->right, &right_min);
1773         min = sval_binop(left_min, '+', right_min);
1774
1775         set_extra_chunk_true_false(left, NULL, alloc_estate_range(min, max));
1776         return 1;
```



```
1777 }
1778
1779 static void match_comparison(struct expression *expr)
1780 {
1781         struct expression *left_orig = strip_parens(expr->left);
1782         struct expression *right_orig = strip_parens(expr->right);
1783         struct expression *left, *right, *tmp;
1784         struct expression *prev;
1785         struct symbol *type;
1786         int redo, count;
1787
1788         if (match_func_comparison(expr))
1789                 return;
1790
1791         type = get_type(expr);
1792         if (!type)
1793                 type = &llong_ctype;
1794
1795         if (handle_integer_overflow_test(expr))
1796                 return;
1797
1798         left = left_orig;
1799         right = right_orig;
1800         move_known_values(&left, &right);
1801         handle_comparison(type, left, expr->op, right);
1802
1803         left = left_orig;
1804         right = right_orig;
1805         if (do_simple_algebra(&left, &right))
1806                 handle_comparison(type, left, expr->op, right);
1807
1808         prev = get_assigned_expr(left_orig);
1809         if (is_simple_math(prev) && has_variable(prev, left_orig) == 0) {
1810                 left = prev;
1811                 right = right_orig;
1812                 move_known_values(&left, &right);
1813                 handle_comparison(type, left, expr->op, right);
1814         }
1815
1816         prev = get_assigned_expr(right_orig);
1817         if (is_simple_math(prev) && has_variable(prev, right_orig) == 0) {
1818                 left = left_orig;
1819                 right = prev;
1820                 move_known_values(&left, &right);
1821                 handle_comparison(type, left, expr->op, right);
1822         }
1823
1824         redo = 0;
1825         left = left_orig;
1826         right = right_orig;
1827         if (get_last_expr_from_expression_stmt(left_orig)) {
1828                 left = get_last_expr_from_expression_stmt(left_orig);
1829                 redo = 1;
1830         }
1831         if (get_last_expr_from_expression_stmt(right_orig)) {
1832                 right = get_last_expr_from_expression_stmt(right_orig);
1833                 redo = 1;
1834         }
1835
1836         if (!redo)
1837                 return;
1838
1839         count = 0;
1840         while ((tmp = get_assigned_expr(left))) {
1841                 if (count++ > 3)
1842                         break;
```

```
1843                    left = strip_expr(tmp);
1844            }
1845            count = 0;
1846            while ((tmp = get_assigned_expr(right))) {
1847                    if (count++ > 3)
1848                            break;
1849                    right = strip_expr(tmp);
1850            }
1851
1852            handle_comparison(type, left, expr->op, right);
1853    }
1854
1855    static sval_t get_high_mask(sval_t known)
1856    {
1857            sval_t ret;
1858            int i;
1859
1860            ret = known;
1861            ret.value = 0;
1862
1863            for (i = type_bits(known.type) - 1; i >= 0; i--) {
1864                    if (known.uvalue & (1ULL << i))
1865                            ret.uvalue |= (1ULL << i);
1866                    else
1867                            return ret;
1868
1869            }
1870            return ret;
1871    }
1872
1873    static void handle_AND_op(struct expression *var, sval_t known)
1874    {
1875            struct range_list *orig_rl;
1876            struct range_list *true_rl = NULL;
1877            struct range_list *false_rl = NULL;
1878            int bit;
1879            sval_t low_mask = known;
1880            sval_t high_mask;
1881            sval_t max;
1882
1883            get_absolute_rl(var, &orig_rl);
1884
1885            if (known.value > 0) {
1886                    bit = ffsll(known.value) - 1;
1887                    low_mask.uvalue = (1ULL << bit) - 1;
1888                    true_rl = remove_range(orig_rl, sval_type_val(known.type, 0), lo
1889            }
1890            high_mask = get_high_mask(known);
1891            if (high_mask.value) {
1892                    bit = ffsll(high_mask.value) - 1;
1893                    low_mask.uvalue = (1ULL << bit) - 1;
1894
1895                    false_rl = orig_rl;
1896                    if (sval_is_negative(rl_min(orig_rl)))
1897                            false_rl = remove_range(false_rl, sval_type_min(known.ty
1898                    false_rl = remove_range(false_rl, low_mask, sval_type_max(known.
1899                    if (type_signed(high_mask.type) && type_unsigned(rl_type(false_r
1900                            false_rl = remove_range(false_rl,
1901                                                    sval_type_val(rl_type(false_rl),
1902                                                  sval_type_val(rl_type(false_rl), -1));
1903                    }
1904            } else if (known.value == 1 &&
1905                       get_hard_max(var, &max) &&
1906                       sval_cmp(max, rl_max(orig_rl)) == 0 &&
1907                       max.value & 1) {
1908                    false_rl = remove_range(orig_rl, max, max);
```



```
1909            }
1910            set_extra_expr_true_false(var,
1911                                      true_rl ? alloc_estate_rl(true_rl) : NULL,
1912                                      false_rl ? alloc_estate_rl(false_rl) : NULL);
1913    }
1914
1915    static void handle_AND_condition(struct expression *expr)
1916    {
1917            sval_t known;
1918
1919            if (get_implied_value(expr->left, &known))
1920                    handle_AND_op(expr->right, known);
1921            else if (get_implied_value(expr->right, &known))
1922                    handle_AND_op(expr->left, known);
1923    }
1924
1925    static void handle_MOD_condition(struct expression *expr)
1926    {
1927            struct range_list *orig_rl;
1928            struct range_list *true_rl;
1929            struct range_list *false_rl = NULL;
1930            sval_t right;
1931            sval_t zero;
1932
1933            if (!get_implied_value(expr->right, &right) || right.value == 0)
1934                    return;
1935            get_absolute_rl(expr->left, &orig_rl);
1936
1937            zero.value = 0;
1938            zero.type = rl_type(orig_rl);
1939
1940            /* We're basically dorking around the min and max here */
1941            true_rl = remove_range(orig_rl, zero, zero);
1942            if (!sval_is_max(rl_max(true_rl)) &&
1943                !(rl_max(true_rl).value % right.value))
1944                    true_rl = remove_range(true_rl, rl_max(true_rl), rl_max(true_rl)
1945
1946            if (rl_equiv(true_rl, orig_rl))
1947                    true_rl = NULL;
1948
1949            if (sval_is_positive(rl_min(orig_rl)) &&
1950                (rl_max(orig_rl).value - rl_min(orig_rl).value) / right.value < 5) {
1951                    sval_t add;
1952                    int i;
1953
1954                    add = rl_min(orig_rl);
1955                    add.value += right.value - (add.value % right.value);
1956                    add.value -= right.value;
1957
1958                    for (i = 0; i < 5; i++) {
1959                            add.value += right.value;
1960                            if (add.value > rl_max(orig_rl).value)
1961                                    break;
1962                            add_range(&false_rl, add, add);
1963                    }
1964            } else {
1965                    if (rl_min(orig_rl).uvalue != 0 &&
1966                        rl_min(orig_rl).uvalue < right.uvalue) {
1967                            sval_t chop = right;
1968                            chop.value--;
1969                            false_rl = remove_range(orig_rl, zero, chop);
1970                    }
1971
1972                    if (!sval_is_max(rl_max(orig_rl)) &&
1973                        (rl_max(orig_rl).value % right.value)) {
1974                            sval_t chop = rl_max(orig_rl);
```

```
1975                        chop.value -= chop.value % right.value;
1976                        chop.value++;
1977                        if (!false_rl)
1978                                false_rl = clone_rl(orig_rl);
1979                        false_rl = remove_range(false_rl, chop, rl_max(orig_rl))
1980                }
1981        }

1983        set_extra_expr_true_false(expr->left,
1984                                  true_rl ? alloc_estate_rl(true_rl) : NULL,
1985                                  false_rl ? alloc_estate_rl(false_rl) : NULL);
1986 }

1988 /* this is actually hooked from smatch_implied.c...  it's hacky, yes */
1989 void __extra_match_condition(struct expression *expr)
1990 {
1991        struct smatch_state *pre_state;
1992        struct smatch_state *true_state;
1993        struct smatch_state *false_state;
1994        struct range_list *pre_rl;

1996        expr = strip_expr(expr);
1997        switch (expr->type) {
1998        case EXPR_CALL:
1999                function_comparison(expr, SPECIAL_NOTEQUAL, zero_expr());
2000                return;
2001        case EXPR_PREOP:
2002        case EXPR_SYMBOL:
2003        case EXPR_DEREF: {
2004                sval_t zero;

2006                zero = sval_blank(expr);
2007                zero.value = 0;

2009                pre_state = get_extra_state(expr);
2010                if (estate_is_empty(pre_state))
2011                        return;
2012                if (pre_state)
2013                        pre_rl = estate_rl(pre_state);
2014                else
2015                        get_absolute_rl(expr, &pre_rl);
2016                if (possibly_true_rl(pre_rl, SPECIAL_EQUAL, rl_zero()))
2017                        false_state = alloc_estate_sval(zero);
2018                else
2019                        false_state = alloc_estate_empty();
2020                true_state = alloc_estate_rl(remove_range(pre_rl, zero, zero));
2021                set_extra_expr_true_false(expr, true_state, false_state);
2022                return;
2023        }
2024        case EXPR_COMPARE:
2025                match_comparison(expr);
2026                return;
2027        case EXPR_ASSIGNMENT:
2028                __extra_match_condition(expr->left);
2029                return;
2030        case EXPR_BINOP:
2031                if (expr->op == '&')
2032                        handle_AND_condition(expr);
2033                if (expr->op == '%')
2034                        handle_MOD_condition(expr);
2035                return;
2036        }
2037 }

2039 static void assume_indexes_are_valid(struct expression *expr)
2040 {
```



```
2041        struct expression *array_expr;
2042        int array_size;
2043        struct expression *offset;
2044        struct symbol *offset_type;
2045        struct range_list *rl_before;
2046        struct range_list *rl_after;
2047        struct range_list *filter = NULL;
2048        sval_t size;

2050        expr = strip_expr(expr);
2051        if (!is_array(expr))
2052                return;

2054        offset = get_array_offset(expr);
2055        offset_type = get_type(offset);
2056        if (offset_type && type_signed(offset_type)) {
2057                filter = alloc_rl(sval_type_min(offset_type),
2058                                  sval_type_val(offset_type, -1));
2059        }

2061        array_expr = get_array_base(expr);
2062        array_size = get_real_array_size(array_expr);
2063        if (array_size > 1) {
2064                size = sval_type_val(offset_type, array_size);
2065                add_range(&filter, size, sval_type_max(offset_type));
2066        }

2068        if (!filter)
2069                return;
2070        get_absolute_rl(offset, &rl_before);
2071        rl_after = rl_filter(rl_before, filter);
2072        if (rl_equiv(rl_before, rl_after))
2073                return;
2074        set_extra_expr_nomod(offset, alloc_estate_rl(rl_after));
2075 }

2077 /* returns 1 if it is not possible for expr to be value, otherwise returns 0 */
2078 int implied_not_equal(struct expression *expr, long long val)
2079 {
2080        return !possibly_false(expr, SPECIAL_NOTEQUAL, value_expr(val));
2081 }

2083 int implied_not_equal_name_sym(char *name, struct symbol *sym, long long val)
2084 {
2085        struct smatch_state *estate;

2087        estate = get_state(SMATCH_EXTRA, name, sym);
2088        if (!estate)
2089                return 0;
2090        if (!rl_has_sval(estate_rl(estate), sval_type_val(estate_type(estate), 0
2091                return 1;
2092        return 0;
2093 }

2095 int parent_is_null_var_sym(const char *name, struct symbol *sym)
2096 {
2097        char buf[256];
2098        char *start;
2099        char *end;
2100        struct smatch_state *state;

2102        strncpy(buf, name, sizeof(buf) - 1);
2103        buf[sizeof(buf) - 1] = '\0';

2105        start = &buf[0];
2106        while (*start == '*') {
```

```
2107                 start++;
2108                 state = get_state(SMATCH_EXTRA, start, sym);
2109                 if (!state)
2110                         continue;
2111                 if (!estate_rl(state))
2112                         return 1;
2113                 if (estate_min(state).value == 0 &&
2114                     estate_max(state).value == 0)
2115                         return 1;
2116         }
2117
2118         start = &buf[0];
2119         while (*start == '&')
2120                 start++;
2121
2122         while ((end = strrchr(start, '-'))) {
2123                 *end = '\0';
2124                 state = __get_state(SMATCH_EXTRA, start, sym);
2125                 if (!state)
2126                         continue;
2127                 if (estate_min(state).value == 0 &&
2128                     estate_max(state).value == 0)
2129                         return 1;
2130         }
2131         return 0;
2132 }
2133
2134 int parent_is_null(struct expression *expr)
2135 {
2136         struct symbol *sym;
2137         char *var;
2138         int ret = 0;
2139
2140         expr = strip_expr(expr);
2141         var = expr_to_var_sym(expr, &sym);
2142         if (!var || !sym)
2143                 goto free;
2144         ret = parent_is_null_var_sym(var, sym);
2145 free:
2146         free_string(var);
2147         return ret;
2148 }
2149
2150 static int param_used_callback(void *found, int argc, char **argv, char **azColN
2151 {
2152         *(int *)found = 1;
2153         return 0;
2154 }
2155
2156 static int filter_unused_kzalloc_info(struct expression *call, int param, char *
2157 {
2158         sval_t sval;
2159         int found = 0;
2160
2161         /* for function pointers assume everything is used */
2162         if (call->fn->type != EXPR_SYMBOL)
2163                 return 0;
2164
2165         /*
2166          * This is to handle __builtin_mul_overflow().  In an ideal world we
2167          * would only need this for invalid code.
2168          *
2169          */
2170         if (!call->fn->symbol)
2171                 return 0;
```



```
2172
2173         /*
2174          * kzalloc() information is treated as special because so there is just
2175          * a lot of stuff initialized to zero and it makes building the database
2176          * take hours and hours.
2177          *
2178          * In theory, we should just remove this line and not pass any unused
2179          * information, but I'm not sure enough that this code works so I want
2180          * to hold off on that for now.
2181          */
2182         if (!estate_get_single_value(sm->state, &sval) || sval.value != 0)
2183                 return 0;
2184
2185         run_sql(&param_used_callback, &found,
2186                 "select * from return_implies where %s and type = %d and paramet
2187                 get_static_filter(call->fn->symbol), PARAM_USED, param, printed_
2188         if (found)
2189                 return 0;
2190
2191         /* If the database is not built yet, then assume everything is used */
2192         run_sql(&param_used_callback, &found,
2193                 "select * from return_implies where %s and type = %d;",
2194                 get_static_filter(call->fn->symbol), PARAM_USED);
2195         if (!found)
2196                 return 0;
2197
2198         return 1;
2199 }
2200
2201 struct range_list *intersect_with_real_abs_var_sym(const char *name, struct symb
2202 {
2203         struct smatch_state *state;
2204
2205         /*
2206          * Here is the difference between implied value and real absolute, say
2207          * you have:
2208          *
2209          *      int a = (u8)x;
2210          *
2211          * Then you know that a is 0-255.  That's real absolute.  But you don't
2212          * know for sure that it actually goes up to 255.  So it's not implied.
2213          * Implied indicates a degree of certainty.
2214          *
2215          * But then say you cap "a" at 8.  That means you know it goes up to
2216          * 8.  So now the implied value is s32min-8.  But you can combine it
2217          * with the real absolute to say that actually it's 0-8.
2218          *
2219          * We are combining it here.  But now that I think about it, this is
2220          * probably not the ideal place to combine it because it should proably
2221          * be done earlier.  Oh well, this is an improvement on what was there
2222          * before so I'm going to commit this code.
2223          *
2224          */
2225
2226         state = get_real_absolute_state_var_sym(name, sym);
2227         if (!state || !estate_rl(state))
2228                 return start;
2229
2230         return rl_intersection(estate_rl(state), start);
2231 }
2232
2233 struct range_list *intersect_with_real_abs_expr(struct expression *expr, struct
2234 {
2235         struct smatch_state *state;
2236         struct range_list *abs_rl;
2237
2238         state = get_real_absolute_state(expr);
```

```
2239         if (!state || !estate_rl(state))
2240                 return start;

2242         abs_rl = cast_rl(rl_type(start), estate_rl(state));
2243         return rl_intersection(abs_rl, start);
2244 }

2246 static void struct_member_callback(struct expression *call, int param, char *pri
2247 {
2248         struct range_list *rl;

2250         if (estate_is_whole(sm->state))
2251                 return;
2252         if (filter_unused_kzalloc_info(call, param, printed_name, sm))
2253                 return;
2254         rl = estate_rl(sm->state);
2255         rl = intersect_with_real_abs_var_sym(sm->name, sm->sym, rl);
2256         sql_insert_caller_info(call, PARAM_VALUE, param, printed_name, show_rl(r
2257         if (estate_has_fuzzy_max(sm->state))
2258                 sql_insert_caller_info(call, FUZZY_MAX, param, printed_name,
2259                                        sval_to_str(estate_get_fuzzy_max(sm->stat
2260 }

2262 static void returned_struct_members(int return_id, char *return_ranges, struct e
2263 {
2264         struct symbol *returned_sym;
2265         struct sm_state *sm;
2266         const char *param_name;
2267         char *compare_str;
2268         char buf[256];

2270         returned_sym = expr_to_sym(expr);
2271         if (!returned_sym)
2272                 return;

2274         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
2275                 if (!estate_rl(sm->state))
2276                         continue;
2277                 if (returned_sym != sm->sym)
2278                         continue;

2280                 param_name = get_param_name(sm);
2281                 if (!param_name)
2282                         continue;
2283                 if (strcmp(param_name, "$") == 0)
2284                         continue;
2285                 compare_str = name_sym_to_param_comparison(sm->name, sm->sym);
2286                 if (!compare_str && estate_is_whole(sm->state))
2287                         continue;
2288                 snprintf(buf, sizeof(buf), "%s%s", sm->state->name, compare_str

2290                 sql_insert_return_states(return_id, return_ranges, PARAM_VALUE,
2291                                          -1, param_name, buf);
2292         } END_FOR_EACH_SM(sm);
2293 }

2295 static void db_limited_before(void)
2296 {
2297         unmatched_stree = clone_stree(__get_cur_stree());
2298 }

2300 static void db_limited_after(void)
2301 {
2302         free_stree(&unmatched_stree);
2303 }
```



```
2305 static int rl_fits_in_type(struct range_list *rl, struct symbol *type)
2306 {
2307         if (type_bits(rl_type(rl)) <= type_bits(type))
2308                 return 1;
2309         if (sval_cmp(rl_max(rl), sval_type_max(type)) > 0)
2310                 return 0;
2311         if (sval_is_negative(rl_min(rl)) &&
2312             sval_cmp(rl_min(rl), sval_type_min(type)) < 0)
2313                 return 0;
2314         return 1;
2315 }

2317 static int basically_the_same(struct range_list *orig, struct range_list *new)
2318 {
2319         if (rl_equiv(orig, new))
2320                 return 1;

2322         /*
2323          * The whole range is essentially the same as 0,4096-27777777777 so
2324          * don't overwrite the implications just to store that.
2325          *
2326          */
2327         if (rl_type(orig)->type == SYM_PTR &&
2328             is_whole_rl(orig) &&
2329             rl_min(new).value == 0 &&
2330             rl_max(new).value == valid_ptr_max)
2331                 return 1;
2332         return 0;
2333 }

2335 static void db_param_limit_binops(struct expression *arg, char *key, struct rang
2336 {
2337         struct range_list *left_rl;
2338         sval_t zero = { .type = rl_type(rl), };
2339         sval_t sval;

2341         if (arg->op != '*')
2342                 return;
2343         if (!get_implied_value(arg->right, &sval))
2344                 return;
2345         if (can_integer_overflow(get_type(arg), arg))
2346                 return;

2348         left_rl = rl_binop(rl, '/', alloc_rl(sval, sval));
2349         if (!rl_has_sval(rl, zero))
2350                 left_rl = remove_range(left_rl, zero, zero);

2352         set_extra_expr_nomod(arg->left, alloc_estate_rl(left_rl));
2353 }

2355 static void db_param_limit_filter(struct expression *expr, int param, char *key,
2356 {
2357         struct expression *arg;
2358         char *name;
2359         struct symbol *sym;
2360         struct var_sym_list *vsl = NULL;
2361         struct sm_state *sm;
2362         struct symbol *compare_type, *var_type;
2363         struct range_list *rl;
2364         struct range_list *limit;
2365         struct range_list *new;
2366         char *tmp_name;
2367         struct symbol *tmp_sym;

2369         while (expr->type == EXPR_ASSIGNMENT)
2370                 expr = strip_expr(expr->right);
```

```
2371         if (expr->type != EXPR_CALL)
2372                 return;
2373
2374         arg = get_argument_from_call_expr(expr->args, param);
2375         if (!arg)
2376                 return;
2377
2378         name = get_chunk_from_key(arg, key, &sym, &vsl);
2379         if (!name)
2380                 return;
2381         if (op != PARAM_LIMIT && !sym)
2382                 goto free;
2383
2384         if (strcmp(key, "$") == 0)
2385                 compare_type = get_arg_type(expr->fn, param);
2386         else
2387                 compare_type = get_member_type_from_key(arg, key);
2388
2389         sm = get_sm_state(SMATCH_EXTRA, name, sym);
2390         if (sm)
2391                 rl = estate_rl(sm->state);
2392         else
2393                 rl = alloc_whole_rl(compare_type);
2394
2395         if (op == PARAM_LIMIT && !rl_fits_in_type(rl, compare_type))
2396                 goto free;
2397
2398         call_results_to_rl(expr, compare_type, value, &limit);
2399         new = rl_intersection(rl, limit);
2400
2401         var_type = get_member_type_from_key(arg, key);
2402         new = cast_rl(var_type, new);
2403
2404         /* We want to preserve the implications here */
2405         if (sm && basically_the_same(estate_rl(sm->state), new))
2406                 goto free;
2407         tmp_name = map_long_to_short_name_sym(name, sym, &tmp_sym);
2408         if (tmp_name && tmp_sym) {
2409                 free_string(name);
2410                 name = tmp_name;
2411                 sym = tmp_sym;
2412         }
2413
2414         if (op == PARAM_LIMIT)
2415                 set_extra_nomod_vsl(name, sym, vsl, NULL, alloc_estate_rl(new));
2416         else
2417                 set_extra_mod(name, sym, NULL, alloc_estate_rl(new));
2418
2419         if (op == PARAM_LIMIT && arg->type == EXPR_BINOP)
2420                 db_param_limit_binops(arg, key, new);
2421 free:
2422         free_string(name);
2423 }
2424
2425 static void db_param_limit(struct expression *expr, int param, char *key, char *
2426 {
2427         db_param_limit_filter(expr, param, key, value, PARAM_LIMIT);
2428 }
2429
2430 static void db_param_filter(struct expression *expr, int param, char *key, char
2431 {
2432         db_param_limit_filter(expr, param, key, value, PARAM_FILTER);
2433 }
2434
2435 static void db_param_add_set(struct expression *expr, int param, char *key, char
2436 {
```



```
2437         struct expression *arg;
2438         char *name, *tmp_name;
2439         struct symbol *sym, *tmp_sym;
2440         struct symbol *param_type, *arg_type;
2441         struct smatch_state *state;
2442         struct range_list *new = NULL;
2443         struct range_list *added = NULL;
2444
2445         while (expr->type == EXPR_ASSIGNMENT)
2446                 expr = strip_expr(expr->right);
2447         if (expr->type != EXPR_CALL)
2448                 return;
2449
2450         arg = get_argument_from_call_expr(expr->args, param);
2451         if (!arg)
2452                 return;
2453
2454         arg_type = get_arg_type_from_key(expr->fn, param, arg, key);
2455         param_type = get_member_type_from_key(arg, key);
2456         name = get_variable_from_key(arg, key, &sym);
2457         if (!name || !sym)
2458                 goto free;
2459
2460         state = get_state(SMATCH_EXTRA, name, sym);
2461         if (state)
2462                 new = estate_rl(state);
2463
2464         call_results_to_rl(expr, arg_type, value, &added);
2465         added = cast_rl(param_type, added);
2466         if (op == PARAM_SET)
2467                 new = added;
2468         else
2469                 new = rl_union(new, added);
2470
2471         tmp_name = map_long_to_short_name_sym_nostack(name, sym, &tmp_sym);
2472         if (tmp_name && tmp_sym) {
2473                 free_string(name);
2474                 name = tmp_name;
2475                 sym = tmp_sym;
2476         }
2477         set_extra_mod(name, sym, NULL, alloc_estate_rl(new));
2478 free:
2479         free_string(name);
2480 }
2481
2482 static void db_param_add(struct expression *expr, int param, char *key, char *va
2483 {
2484         in_param_set = true;
2485         db_param_add_set(expr, param, key, value, PARAM_ADD);
2486         in_param_set = false;
2487 }
2488
2489 static void db_param_set(struct expression *expr, int param, char *key, char *va
2490 {
2491         in_param_set = true;
2492         db_param_add_set(expr, param, key, value, PARAM_SET);
2493         in_param_set = false;
2494 }
2495
2496 static void db_param_value(struct expression *expr, int param, char *key, char *
2497 {
2498         struct expression *call;
2499         char *name;
2500         struct symbol *sym;
2501         struct symbol *type;
2502         struct range_list *rl = NULL;
```

```
2504         if (param != -1)
2505                 return;
2506
2507         call = expr;
2508         while (call->type == EXPR_ASSIGNMENT)
2509                 call = strip_expr(call->right);
2510         if (call->type != EXPR_CALL)
2511                 return;
2512
2513         type = get_member_type_from_key(expr->left, key);
2514         name = get_variable_from_key(expr->left, key, &sym);
2515         if (!name || !sym)
2516                 goto free;
2517
2518         call_results_to_rl(call, type, value, &rl);
2519
2520         set_extra_mod(name, sym, NULL, alloc_estate_rl(rl));
2521 free:
2522         free_string(name);
2523 }
2524
2525 static void match_call_info(struct expression *expr)
2526 {
2527         struct smatch_state *state;
2528         struct range_list *rl = NULL;
2529         struct expression *arg;
2530         struct symbol *type;
2531         int i = 0;
2532
2533         FOR_EACH_PTR(expr->args, arg) {
2534                 type = get_arg_type(expr->fn, i);
2535
2536                 get_absolute_rl(arg, &rl);
2537                 rl = cast_rl(type, rl);
2538
2539                 if (!is_whole_rl(rl)) {
2540                         rl = intersect_with_real_abs_expr(arg, rl);
2541                         sql_insert_caller_info(expr, PARAM_VALUE, i, "$", show_r
2542                 }
2543                 state = get_state_expr(SMATCH_EXTRA, arg);
2544                 if (estate_has_fuzzy_max(state)) {
2545                         sql_insert_caller_info(expr, FUZZY_MAX, i, "$",
2546                                                sval_to_str(estate_get_fuzzy_max(
2547                 }
2548                 i++;
2549         } END_FOR_EACH_PTR(arg);
2550 }
2551
2552 static void set_param_value(const char *name, struct symbol *sym, char *key, cha
2553 {
2554         struct range_list *rl = NULL;
2555         struct smatch_state *state;
2556         struct symbol *type;
2557         char fullname[256];
2558         sval_t dummy;
2559
2560         if (strcmp(key, "*$") == 0)
2561                 snprintf(fullname, sizeof(fullname), "*%s", name);
2562         else if (strncmp(key, "$", 1) == 0)
2563                 snprintf(fullname, 256, "%s%s", name, key + 1);
2564         else
2565                 return;
2566
2567         type = get_member_type_from_key(symbol_expression(sym), key);
2568         str_to_rl(type, value, &rl);
```



```
2569         state = alloc_estate_rl(rl);
2570         if (estate_get_single_value(state, &dummy))
2571                 estate_set_hard_max(state);
2572         set_state(SMATCH_EXTRA, fullname, sym, state);
2573 }
2574
2575 static void set_param_hard_max(const char *name, struct symbol *sym, char *key,
2576 {
2577         struct range_list *rl = NULL;
2578         struct smatch_state *state;
2579         struct symbol *type;
2580         char fullname[256];
2581         sval_t max;
2582
2583         if (strcmp(key, "*$") == 0)
2584                 snprintf(fullname, sizeof(fullname), "*%s", name);
2585         else if (strncmp(key, "$", 1) == 0)
2586                 snprintf(fullname, 256, "%s%s", name, key + 1);
2587         else
2588                 return;
2589
2590         state = get_state(SMATCH_EXTRA, fullname, sym);
2591         if (!state)
2592                 return;
2593         type = get_member_type_from_key(symbol_expression(sym), key);
2594         str_to_rl(type, value, &rl);
2595         if (!rl_to_sval(rl, &max))
2596                 return;
2597         estate_set_fuzzy_max(state, max);
2598 }
2599
2600 struct sm_state *get_extra_sm_state(struct expression *expr)
2601 {
2602         char *name;
2603         struct symbol *sym;
2604         struct sm_state *ret = NULL;
2605
2606         name = expr_to_known_chunk_sym(expr, &sym);
2607         if (!name)
2608                 goto free;
2609
2610         ret = get_sm_state(SMATCH_EXTRA, name, sym);
2611 free:
2612         free_string(name);
2613         return ret;
2614 }
2615
2616 struct smatch_state *get_extra_state(struct expression *expr)
2617 {
2618         struct sm_state *sm;
2619
2620         sm = get_extra_sm_state(expr);
2621         if (!sm)
2622                 return NULL;
2623         return sm->state;
2624 }
2625
2626 void register_smatch_extra(int id)
2627 {
2628         my_id = id;
2629
2630         add_merge_hook(my_id, &merge_estates);
2631         add_unmatched_state_hook(my_id, &unmatched_state);
2632         select_caller_info_hook(set_param_value, PARAM_VALUE);
2633         select_caller_info_hook(set_param_hard_max, FUZZY_MAX);
2634         select_return_states_before(&db_limited_before);
```

```
2635         select_return_states_hook(PARAM_LIMIT, &db_param_limit);
2636         select_return_states_hook(PARAM_FILTER, &db_param_filter);
2637         select_return_states_hook(PARAM_ADD, &db_param_add);
2638         select_return_states_hook(PARAM_SET, &db_param_set);
2639         select_return_states_hook(PARAM_VALUE, &db_param_value);
2640         select_return_states_after(&db_limited_after);
2641 }

2643 static void match_link_modify(struct sm_state *sm, struct expression *mod_expr)
2644 {
2645         struct var_sym_list *links;
2646         struct var_sym *tmp;
2647         struct smatch_state *state;

2649         links = sm->state->data;

2651         FOR_EACH_PTR(links, tmp) {
2652                 if (sm->sym == tmp->sym &&
2653                     strcmp(sm->name, tmp->var) == 0)
2654                         continue;
2655                 state = get_state(SMATCH_EXTRA, tmp->var, tmp->sym);
2656                 if (!state)
2657                         continue;
2658                 set_state(SMATCH_EXTRA, tmp->var, tmp->sym, alloc_estate_whole(e
2659         } END_FOR_EACH_PTR(tmp);
2660         set_state(link_id, sm->name, sm->sym, &undefined);
2661 }

2663 void register_smatch_extra_links(int id)
2664 {
2665         link_id = id;
2666 }

2668 void register_smatch_extra_late(int id)
2669 {
2670         add_merge_hook(link_id, &merge_link_states);
2671         add_modification_hook(link_id, &match_link_modify);
2672         add_hook(&match_dereferences, DEREF_HOOK);
2673         add_hook(&match_pointer_as_array, OP_HOOK);
2674         select_return_implies_hook(DEREFERENCE, &set_param_dereferenced);
2675         add_hook(&match_function_call, FUNCTION_CALL_HOOK);
2676         add_hook(&match_assign, ASSIGNMENT_HOOK);
2677         add_hook(&match_assign, GLOBAL_ASSIGNMENT_HOOK);
2678         add_hook(&unop_expr, OP_HOOK);
2679         add_hook(&asm_expr, ASM_HOOK);

2681         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
2682         add_member_info_callback(my_id, struct_member_callback);
2683         add_split_return_callback(&returned_struct_members);

2685 //      add_hook(&assume_indexes_are_valid, OP_HOOK);
2686 }
```

```
**********************************************************
   11644 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_extra.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

DECLARE_PTR_LIST(range_list, struct data_range);
DECLARE_PTR_LIST(range_list_stack, struct range_list);

struct relation {
        char *name;
        struct symbol *sym;
};

DECLARE_PTR_LIST(related_list, struct relation);

struct data_info {
        struct related_list *related;
        struct range_list *value_ranges;
        sval_t fuzzy_max;
        unsigned int hard_max:1;
};
DECLARE_ALLOCATOR(data_info);

extern struct string_list *__ignored_macros;

/* these are implemented in smatch_ranges.c */
struct range_list *rl_zero(void);
struct range_list *rl_one(void);
char *show_rl(struct range_list *list);
int str_to_comparison_arg(const char *c, struct expression *call, int *compariso
void str_to_rl(struct symbol *type, char *value, struct range_list **rl);
void call_results_to_rl(struct expression *call, struct symbol *type, char *valu

struct data_range *alloc_range(sval_t min, sval_t max);
struct data_range *alloc_range_perm(sval_t min, sval_t max);

struct range_list *alloc_rl(sval_t min, sval_t max);
struct range_list *clone_rl(struct range_list *list);
struct range_list *clone_rl_permanent(struct range_list *list);
struct range_list *alloc_whole_rl(struct symbol *type);

void add_range(struct range_list **list, sval_t min, sval_t max);
struct range_list *remove_range(struct range_list *list, sval_t min, sval_t max)
void tack_on(struct range_list **list, struct data_range *drange);

int true_comparison_range(struct data_range *left, int comparison, struct data_r
int true_comparison_range_LR(int comparison, struct data_range *var, struct data
int false_comparison_range_LR(int comparison, struct data_range *var, struct dat
```



```
int possibly_true(struct expression *left, int comparison, struct expression *ri
int possibly_true_rl(struct range_list *left_ranges, int comparison, struct rang
int possibly_true_rl_LR(int comparison, struct range_list *a, struct range_list

int possibly_false(struct expression *left, int comparison, struct expression *r
int possibly_false_rl(struct range_list *left_ranges, int comparison, struct ran
int possibly_false_rl_LR(int comparison, struct range_list *a, struct range_list

int rl_has_sval(struct range_list *rl, sval_t sval);
int ranges_equiv(struct data_range *one, struct data_range *two);

int rl_equiv(struct range_list *one, struct range_list *two);
int is_whole_rl(struct range_list *rl);
int is_whole_rl_non_zero(struct range_list *rl);
int estate_is_unknown(struct smatch_state *state);

sval_t rl_min(struct range_list *rl);
sval_t rl_max(struct range_list *rl);
int rl_to_sval(struct range_list *rl, sval_t *sval);
struct symbol *rl_type(struct range_list *rl);

struct range_list *rl_invert(struct range_list *orig);
struct range_list *rl_filter(struct range_list *rl, struct range_list *filter);
struct range_list *rl_intersection(struct range_list *one, struct range_list *tw
struct range_list *rl_union(struct range_list *one, struct range_list *two);
struct range_list *rl_binop(struct range_list *left, int op, struct range_list *

void push_rl(struct range_list_stack **rl_stack, struct range_list *rl);
struct range_list *pop_rl(struct range_list_stack **rl_stack);
struct range_list *top_rl(struct range_list_stack *rl_stack);
void filter_top_rl(struct range_list_stack **rl_stack, struct range_list *filter

struct range_list *rl_truncate_cast(struct symbol *type, struct range_list *rl);
struct range_list *cast_rl(struct symbol *type, struct range_list *rl);
int get_implied_rl(struct expression *expr, struct range_list **rl);
int get_absolute_rl(struct expression *expr, struct range_list **rl);
int get_real_absolute_rl(struct expression *expr, struct range_list **rl);
struct range_list *var_to_absolute_rl(struct expression *expr);
int custom_get_absolute_rl(struct expression *expr,
                           struct range_list *(*fn)(struct expression *expr),
                           struct range_list **rl);
int get_implied_rl_var_sym(const char *var, struct symbol *sym, struct range_lis
void split_comparison_rl(struct range_list *left_orig, int op, struct range_list
                struct range_list **left_true_rl, struct range_list **left_false
                struct range_list **right_true_rl, struct range_list **right_fal

void free_data_info_allocs(void);
void free_all_rl(void);

/* smatch_estate.c */

struct smatch_state *alloc_estate_empty(void);
struct smatch_state *alloc_estate_sval(sval_t sval);
struct smatch_state *alloc_estate_range(sval_t min, sval_t max);
struct smatch_state *alloc_estate_rl(struct range_list *rl);
struct smatch_state *alloc_estate_whole(struct symbol *type);
struct smatch_state *clone_estate(struct smatch_state *state);
struct smatch_state *clone_estate_cast(struct symbol *type, struct smatch_state

struct smatch_state *merge_estates(struct smatch_state *s1, struct smatch_state

int estates_equiv(struct smatch_state *one, struct smatch_state *two);
int estate_is_whole(struct smatch_state *state);
int estate_is_empty(struct smatch_state *state);
```

```
127 struct range_list *estate_rl(struct smatch_state *state);
128 struct related_list *estate_related(struct smatch_state *state);

130 sval_t estate_min(struct smatch_state *state);
131 sval_t estate_max(struct smatch_state *state);
132 struct symbol *estate_type(struct smatch_state *state);

134 int estate_has_fuzzy_max(struct smatch_state *state);
135 sval_t estate_get_fuzzy_max(struct smatch_state *state);
136 void estate_set_fuzzy_max(struct smatch_state *state, sval_t max);
137 void estate_copy_fuzzy_max(struct smatch_state *new, struct smatch_state *old);
138 void estate_clear_fuzzy_max(struct smatch_state *state);
139 int estate_has_hard_max(struct smatch_state *state);
140 void estate_set_hard_max(struct smatch_state *state);
141 void estate_clear_hard_max(struct smatch_state *state);
142 int estate_get_hard_max(struct smatch_state *state, sval_t *sval);

144 int estate_get_single_value(struct smatch_state *state, sval_t *sval);
145 struct smatch_state *get_implied_estate(struct expression *expr);

147 struct smatch_state *estate_filter_sval(struct smatch_state *orig, sval_t filter
148 struct smatch_state *estate_filter_range(struct smatch_state *orig, sval_t filte
149 struct data_info *clone_dinfo_perm(struct data_info *dinfo);
150 struct smatch_state *clone_estate_perm(struct smatch_state *state);

152 /* smatch_extra.c */
153 bool is_impossible_variable(struct expression *expr);
154 struct sm_state *get_extra_sm_state(struct expression *expr);
155 struct smatch_state *get_extra_state(struct expression *expr);
156 void call_extra_mod_hooks(const char *name, struct symbol *sym, struct expressio
157 void set_extra_mod(const char *name, struct symbol *sym, struct expression *expr
158 void set_extra_expr_mod(struct expression *expr, struct smatch_state *state);
159 void set_extra_nomod(const char *name, struct symbol *sym, struct expression *ex
160 void set_extra_nomod_vsl(const char *name, struct symbol *sym, struct var_sym_li
161 void set_extra_expr_nomod(struct expression *expr, struct smatch_state *state);
162 void set_extra_mod_helper(const char *name, struct symbol *sym, struct expressio

164 struct data_info *get_dinfo(struct smatch_state *state);

166 void add_extra_mod_hook(void (*fn)(const char *name, struct symbol *sym, struct
167 void add_extra_nomod_hook(void (*fn)(const char *name, struct symbol *sym, struc
168 int implied_not_equal(struct expression *expr, long long val);
169 int implied_not_equal_name_sym(char *name, struct symbol *sym, long long val);
170 int parent_is_null_var_sym(const char *name, struct symbol *sym);
171 int parent_is_null(struct expression *expr);
172 int parent_is_free_var_sym_strict(const char *name, struct symbol *sym);
173 int parent_is_free_var_sym(const char *name, struct symbol *sym);
174 int parent_is_free(struct expression *expr);

176 struct sm_state *__extra_handle_canonical_loops(struct statement *loop, struct s
177 int __iterator_unchanged(struct sm_state *sm);
178 void __extra_pre_loop_hook_after(struct sm_state *sm,
179                                  struct statement *iterator,
180                                  struct expression *condition);

182 /* smatch_equiv.c */
183 void set_equiv(struct expression *left, struct expression *right);
184 void set_related(struct smatch_state *estate, struct related_list *rlist);
185 struct related_list *get_shared_relations(struct related_list *one,
186                                           struct related_list *two);
187 struct related_list *clone_related_list(struct related_list *related);
188 void remove_from_equiv(const char *name, struct symbol *sym);
189 void remove_from_equiv_expr(struct expression *expr);
190 void set_equiv_state_expr(int id, struct expression *expr, struct smatch_state *

192 /* smatch_function_hooks.c */
```



```
193 void function_comparison(struct expression *left, int comparison, struct express

195 /* smatch_expressions.c */
196 struct expression *zero_expr();
197 struct expression *value_expr(long long val);
198 struct expression *member_expression(struct expression *deref, int op, struct id
199 struct expression *preop_expression(struct expression *expr, int op);
200 struct expression *deref_expression(struct expression *expr);
201 struct expression *assign_expression(struct expression *left, int op, struct exp
202 struct expression *binop_expression(struct expression *left, int op, struct expr
203 struct expression *array_element_expression(struct expression *array, struct exp
204 struct expression *symbol_expression(struct symbol *sym);
205 struct expression *string_expression(char *str);
206 struct expression *compare_expression(struct expression *left, int op, struct ex
207 struct expression *unknown_value_expression(struct expression *expr);
208 int is_fake_call(struct expression *expr);
209 struct expression *gen_expression_from_key(struct expression *arg, const char *k
210 void free_tmp_expressions(void);
211 void expr_set_parent_expr(struct expression *expr, struct expression *parent);
212 void expr_set_parent_stmt(struct expression *expr, struct statement *parent);
213 struct expression *expr_get_parent_expr(struct expression *expr);
214 struct statement *expr_get_parent_stmt(struct expression *expr);

216 /* smatch_param_limit.c */
217 struct smatch_state *get_orig_estate(const char *name, struct symbol *sym);

219 /* smatch_real_absolute.c */
220 struct smatch_state *get_real_absolute_state(struct expression *expr);
221 struct smatch_state *get_real_absolute_state_var_sym(const char *name, struct sy

223 /* smatch_imaginary_absolute.c */
224 void __save_imaginary_state(struct expression *expr, struct range_list *true_rl,
225 int get_imaginary_absolute(struct expression *expr, struct range_list **rl);
```

```
**********************************************************
    1508 Fri Dec 21 15:00:25 2018
new/usr/src/tools/smatch/src/smatch_files.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include "parse.h"
#include "smatch.h"

int open_data_file(const char *filename)
{
        char buf[256];
        int fd;

        fd = open(filename, O_RDONLY);
        if (fd >= 0)
                return fd;
        if (!data_dir)
                return -1;
        snprintf(buf, 256, "%s/%s", data_dir, filename);
        return open(buf, O_RDONLY);
}

int open_schema_file(const char *schema)
{
        char buf[256];
        int fd;

        fd = open_data_file(schema);
        if (fd >= 0)
                return fd;
        snprintf(buf, 256, "%s/smatch_data/%s", bin_dir, schema);
        return open(buf, O_RDONLY);
}

struct token *get_tokens_file(const char *filename)
{
        int fd;
        struct token *token;

        if (option_no_data)
                return NULL;
        fd = open_data_file(filename);
        if (fd < 0)
                return NULL;
        token = tokenize(filename, fd, NULL, NULL);
```



```
        close(fd);
        return token;
}
```

```
**********************************************************
   47438 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_flow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2006,2008 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#define _GNU_SOURCE 1
#include <unistd.h>
#include <stdio.h>
#include "token.h"
#include "scope.h"
#include "smatch.h"
#include "smatch_expression_stacks.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

int __in_fake_assign;
int __in_fake_struct_assign;
int in_fake_env;
int final_pass;
int __inline_call;
struct expression  *__inline_fn;

static int __smatch_lineno = 0;

static char *base_file;
static const char *filename;
static char *pathname;
static char *full_filename;
static char *full_base_file;
static char *cur_func;
static unsigned int loop_count;
static int last_goto_statement_handled;
int __expr_stmt_count;
int __in_function_def;
static struct expression_list *switch_expr_stack = NULL;
static struct expression_list *post_op_stack = NULL;

static struct ptr_list *backup;

struct expression_list *big_expression_stack;
struct statement_list *big_statement_stack;
struct statement *__prev_stmt;
struct statement *__cur_stmt;
struct statement *__next_stmt;
int __in_pre_condition = 0;
int __bail_on_rest_of_function = 0;
static struct timeval fn_start_time;
static struct timeval outer_fn_start_time;
```



```
char *get_function(void) { return cur_func; }
int get_lineno(void) { return __smatch_lineno; }
int inside_loop(void) { return !!loop_count; }
int definitely_inside_loop(void) { return !!(loop_count & ~0x08000000); }
struct expression *get_switch_expr(void) { return top_expression(switch_expr_sta
int in_expression_statement(void) { return !!__expr_stmt_count; }

static void split_symlist(struct symbol_list *sym_list);
static void split_declaration(struct symbol_list *sym_list);
static void split_expr_list(struct expression_list *expr_list, struct expression
static void add_inline_function(struct symbol *sym);
static void parse_inline(struct expression *expr);

int option_assume_loops = 0;
int option_two_passes = 0;
struct symbol *cur_func_sym = NULL;
struct stree *global_states;

long long valid_ptr_min = 4096;
long long valid_ptr_max = 2117777777;
sval_t valid_ptr_min_sval = {
        .type = &ptr_ctype,
        {.value = 4096},
};
sval_t valid_ptr_max_sval = {
        .type = &ptr_ctype,
        {.value = LONG_MAX - 100000},
};
struct range_list *valid_ptr_rl;

static void set_valid_ptr_max(void)
{
        if (type_bits(&ptr_ctype) == 32)
                valid_ptr_max = 2117777777;
        else if (type_bits(&ptr_ctype) == 64)
                valid_ptr_max = 211777777777777777LL;

        valid_ptr_max_sval.value = valid_ptr_max;
}

static void alloc_valid_ptr_rl(void)
{
        valid_ptr_rl = alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
        valid_ptr_rl = cast_rl(&ptr_ctype, valid_ptr_rl);
        valid_ptr_rl = clone_rl_permanent(valid_ptr_rl);
}

int outside_of_function(void)
{
        return cur_func_sym == NULL;
}

const char *get_filename(void)
{
        if (option_info && option_full_path)
                return full_base_file;
        if (option_info)
                return base_file;
        if (option_full_path)
                return full_filename;
        return filename;
}

const char *get_base_file(void)
{
        if (option_full_path)
```

```
127                 return full_base_file;
128         return base_file;
129 }

131 static void set_position(struct position pos)
132 {
133         int len;
134         static int prev_stream = -1;

136         if (in_fake_env)
137                 return;

139         if (pos.stream == 0 && pos.line == 0)
140                 return;

142         __smatch_lineno = pos.line;

144         if (pos.stream == prev_stream)
145                 return;

147         filename = stream_name(pos.stream);

149         free(full_filename);
150         pathname = getcwd(NULL, 0);
151         if (pathname) {
152                 len = strlen(pathname) + 1 + strlen(filename) + 1;
153                 full_filename = malloc(len);
154                 snprintf(full_filename, len, "%s/%s", pathname, filename);
155         } else {
156                 full_filename = alloc_string(filename);
157         }
158         free(pathname);
159 }

161 int is_assigned_call(struct expression *expr)
162 {
163         struct expression *parent = expr_get_parent_expr(expr);

165         if (parent &&
166             parent->type == EXPR_ASSIGNMENT &&
167             parent->op == '=' &&
168             strip_expr(parent->right) == expr)
169                 return 1;

171         return 0;
172 }

174 static int is_inline_func(struct expression *expr)
175 {
176         if (expr->type != EXPR_SYMBOL || !expr->symbol)
177                 return 0;
178         if (expr->symbol->ctype.modifiers & MOD_INLINE)
179                 return 1;
180         return 0;
181 }

183 static int is_noreturn_func(struct expression *expr)
184 {
185         if (expr->type != EXPR_SYMBOL || !expr->symbol)
186                 return 0;
187         if (expr->symbol->ctype.modifiers & MOD_NORETURN)
188                 return 1;
189         return 0;
190 }

192 static int inline_budget = 20;
```



```
194 int inlinable(struct expression *expr)
195 {
196         struct symbol *sym;
197         struct statement *last_stmt = NULL;

199         if (__inline_fn)  /* don't nest */
200                 return 0;

202         if (expr->type != EXPR_SYMBOL || !expr->symbol)
203                 return 0;
204         if (is_no_inline_function(expr->symbol->ident->name))
205                 return 0;
206         sym = get_base_type(expr->symbol);
207         if (sym->stmt && sym->stmt->type == STMT_COMPOUND) {
208                 if (ptr_list_size((struct ptr_list *)sym->stmt->stmts) > 10)
209                         return 0;
210                 if (sym->stmt->type != STMT_COMPOUND)
211                         return 0;
212                 last_stmt = last_ptr_list((struct ptr_list *)sym->stmt->stmts);
213         }
214         if (sym->inline_stmt && sym->inline_stmt->type == STMT_COMPOUND) {
215                 if (ptr_list_size((struct ptr_list *)sym->inline_stmt->stmts) >
216                         return 0;
217                 if (sym->inline_stmt->type != STMT_COMPOUND)
218                         return 0;
219                 last_stmt = last_ptr_list((struct ptr_list *)sym->inline_stmt->s
220         }

222         if (!last_stmt)
223                 return 0;

225         /* the magic numbers in this function are pulled out of my bum. */
226         if (last_stmt->pos.line > sym->pos.line + inline_budget)
227                 return 0;

229         return 1;
230 }

232 void __process_post_op_stack(void)
233 {
234         struct expression *expr;

236         FOR_EACH_PTR(post_op_stack, expr) {
237                 __pass_to_client(expr, OP_HOOK);
238         } END_FOR_EACH_PTR(expr);

240         __free_ptr_list((struct ptr_list **)&post_op_stack);
241 }

243 static int handle_comma_assigns(struct expression *expr)
244 {
245         struct expression *right;
246         struct expression *assign;

248         right = strip_expr(expr->right);
249         if (right->type != EXPR_COMMA)
250                 return 0;

252         __split_expr(right->left);
253         __process_post_op_stack();

255         assign = assign_expression(expr->left, '=', right->right);
256         __split_expr(assign);

258         return 1;
```

```
}

/* This is to handle *p++ = foo; assignments */
static int handle_postop_assigns(struct expression *expr)
{
        struct expression *left, *fake_left;
        struct expression *assign;

        left = strip_expr(expr->left);
        if (left->type != EXPR_PREOP || left->op != '*')
                return 0;
        left = strip_expr(left->unop);
        if (left->type != EXPR_POSTOP)
                return 0;

        fake_left = deref_expression(strip_expr(left->unop));
        assign = assign_expression(fake_left, '=', expr->right);

        __split_expr(assign);
        __split_expr(expr->left);

        return 1;
}

static int prev_expression_is_getting_address(struct expression *expr)
{
        struct expression *parent;

        do {
                parent = expr_get_parent_expr(expr);

                if (!parent)
                        return 0;
                if (parent->type == EXPR_PREOP && parent->op == '&')
                        return 1;
                if (parent->type == EXPR_PREOP && parent->op == '(')
                        goto next;
                if (parent->type == EXPR_DEREF && parent->op == '.')
                        goto next;

                return 0;
next:
                expr = parent;
        } while (1);
}

static void handle_builtin_overflow_func(struct expression *expr)
{
        struct expression *a, *b, *res, *assign;
        int op;

        if (sym_name_is("__builtin_add_overflow", expr->fn))
                op = '+';
        else if (sym_name_is("__builtin_sub_overflow", expr->fn))
                op = '-';
        else if (sym_name_is("__builtin_mul_overflow", expr->fn))
                op = '*';
        else
                return;

        a = get_argument_from_call_expr(expr->args, 0);
        b = get_argument_from_call_expr(expr->args, 1);
        res = get_argument_from_call_expr(expr->args, 2);

        assign = assign_expression(deref_expression(res), '=', binop_expression(
        __split_expr(assign);
```



```
}

static int handle__builtin_choose_expr(struct expression *expr)
{
        struct expression *const_expr, *expr1, *expr2;
        sval_t sval;

        if (!sym_name_is("__builtin_choose_expr", expr->fn))
                return 0;

        const_expr = get_argument_from_call_expr(expr->args, 0);
        expr1 = get_argument_from_call_expr(expr->args, 1);
        expr2 = get_argument_from_call_expr(expr->args, 2);

        if (!get_value(const_expr, &sval) || !expr1 || !expr2)
                return 0;
        if (sval.value)
                __split_expr(expr1);
        else
                __split_expr(expr2);
        return 1;
}

static int handle__builtin_choose_expr_assigns(struct expression *expr)
{
        struct expression *const_expr, *right, *expr1, *expr2, *fake;
        sval_t sval;

        right = strip_expr(expr->right);
        if (right->type != EXPR_CALL)
                return 0;
        if (!sym_name_is("__builtin_choose_expr", right->fn))
                return 0;

        const_expr = get_argument_from_call_expr(right->args, 0);
        expr1 = get_argument_from_call_expr(right->args, 1);
        expr2 = get_argument_from_call_expr(right->args, 2);

        if (!get_value(const_expr, &sval) || !expr1 || !expr2)
                return 0;

        fake = assign_expression(expr->left, '=', sval.value ? expr1 : expr2);
        __split_expr(fake);
        return 1;
}

void __split_expr(struct expression *expr)
{
        if (!expr)
                return;

        // sm_msg(" Debug expr_type %d %s", expr->type, show_special(expr->op));

        if (__in_fake_assign && expr->type != EXPR_ASSIGNMENT)
                return;
        if (__in_fake_assign >= 4)  /* don't allow too much nesting */
                return;

        push_expression(&big_expression_stack, expr);
        set_position(expr->pos);
        __pass_to_client(expr, EXPR_HOOK);

        switch (expr->type) {
        case EXPR_PREOP:
                expr_set_parent_expr(expr->unop, expr);
```

```
391                 if (expr->op == '*' &&
392                     !prev_expression_is_getting_address(expr))
393                         __pass_to_client(expr, DEREF_HOOK);
394                 __split_expr(expr->unop);
395                 __pass_to_client(expr, OP_HOOK);
396                 break;
397         case EXPR_POSTOP:
398                 expr_set_parent_expr(expr->unop, expr);
399
400                 __split_expr(expr->unop);
401                 push_expression(&post_op_stack, expr);
402                 break;
403         case EXPR_STATEMENT:
404                 __expr_stmt_count++;
405                 if (expr->statement && !expr->statement) {
406                         stmt_set_parent_stmt(expr->statement,
407                                              last_ptr_list((struct ptr_list *)big_sta
408                 }
409                 __split_stmt(expr->statement);
410                 __expr_stmt_count--;
411                 break;
412         case EXPR_LOGICAL:
413         case EXPR_COMPARE:
414                 expr_set_parent_expr(expr->left, expr);
415                 expr_set_parent_expr(expr->right, expr);
416
417                 __pass_to_client(expr, LOGIC_HOOK);
418                 __handle_logic(expr);
419                 break;
420         case EXPR_BINOP:
421                 expr_set_parent_expr(expr->left, expr);
422                 expr_set_parent_expr(expr->right, expr);
423
424                 __pass_to_client(expr, BINOP_HOOK);
425         case EXPR_COMMA:
426                 expr_set_parent_expr(expr->left, expr);
427                 expr_set_parent_expr(expr->right, expr);
428
429                 __split_expr(expr->left);
430                 __process_post_op_stack();
431                 __split_expr(expr->right);
432                 break;
433         case EXPR_ASSIGNMENT: {
434                 struct expression *right;
435
436                 expr_set_parent_expr(expr->left, expr);
437                 expr_set_parent_expr(expr->right, expr);
438
439                 right = strip_expr(expr->right);
440                 if (!right)
441                         break;
442
443                 __pass_to_client(expr, RAW_ASSIGNMENT_HOOK);
444
445                 /* foo = !bar() */
446                 if (__handle_condition_assigns(expr))
447                         break;
448                 /* foo = (x < 5 ? foo : 5); */
449                 if (__handle_select_assigns(expr))
450                         break;
451                 /* foo = ({frob(); frob(); frob(); 1;}) */
452                 if (__handle_expr_statement_assigns(expr))
453                         break;
454                 /* foo = (3, 4); */
455                 if (handle_comma_assigns(expr))
456                         break;
```



```
457                 if (handle_postop_assigns(expr))
458                         break;
459                 if (handle__builtin_choose_expr_assigns(expr))
460                         break;
461
462                 __split_expr(expr->right);
463                 if (outside_of_function())
464                         __pass_to_client(expr, GLOBAL_ASSIGNMENT_HOOK);
465                 else
466                         __pass_to_client(expr, ASSIGNMENT_HOOK);
467
468                 __fake_struct_member_assignments(expr);
469
470                 if (expr->op == '=' && right->type == EXPR_CALL)
471                         __pass_to_client(expr, CALL_ASSIGNMENT_HOOK);
472
473                 if (get_macro_name(right->pos) &&
474                     get_macro_name(expr->pos) != get_macro_name(right->pos))
475                         __pass_to_client(expr, MACRO_ASSIGNMENT_HOOK);
476
477                 __pass_to_client(expr, ASSIGNMENT_HOOK_AFTER);
478
479                 __split_expr(expr->left);
480                 break;
481         }
482         case EXPR_DEREF:
483                 expr_set_parent_expr(expr->deref, expr);
484
485                 __pass_to_client(expr, DEREF_HOOK);
486                 __split_expr(expr->deref);
487                 break;
488         case EXPR_SLICE:
489                 expr_set_parent_expr(expr->base, expr);
490
491                 __split_expr(expr->base);
492                 break;
493         case EXPR_CAST:
494         case EXPR_FORCE_CAST:
495                 expr_set_parent_expr(expr->cast_expression, expr);
496
497                 __pass_to_client(expr, CAST_HOOK);
498                 __split_expr(expr->cast_expression);
499                 break;
500         case EXPR_SIZEOF:
501                 if (expr->cast_expression)
502                         __pass_to_client(strip_parens(expr->cast_expression),
503                                          SIZEOF_HOOK);
504                 break;
505         case EXPR_OFFSETOF:
506         case EXPR_ALIGNOF:
507                 evaluate_expression(expr);
508                 break;
509         case EXPR_CONDITIONAL:
510         case EXPR_SELECT:
511                 expr_set_parent_expr(expr->conditional, expr);
512                 expr_set_parent_expr(expr->cond_true, expr);
513                 expr_set_parent_expr(expr->cond_false, expr);
514
515                 if (known_condition_true(expr->conditional)) {
516                         __split_expr(expr->cond_true);
517                         break;
518                 }
519                 if (known_condition_false(expr->conditional)) {
520                         __split_expr(expr->cond_false);
521                         break;
522                 }
```

```
523                 __pass_to_client(expr, SELECT_HOOK);
524                 __split_whole_condition(expr->conditional);
525                 __split_expr(expr->cond_true);
526                 __push_true_states();
527                 __use_false_states();
528                 __split_expr(expr->cond_false);
529                 __merge_true_states();
530                 break;
531         case EXPR_CALL:
532                 expr_set_parent_expr(expr->fn, expr);

534                 if (sym_name_is("__builtin_constant_p", expr->fn))
535                         break;
536                 if (handle__builtin_choose_expr(expr))
537                         break;
538                 split_expr_list(expr->args, expr);
539                 __split_expr(expr->fn);
540                 if (is_inline_func(expr->fn))
541                         add_inline_function(expr->fn->symbol);
542                 if (inlinable(expr->fn))
543                         __inline_call = 1;
544                 __process_post_op_stack();
545                 __pass_to_client(expr, FUNCTION_CALL_HOOK_BEFORE);
546                 __pass_to_client(expr, FUNCTION_CALL_HOOK);
547                 __inline_call = 0;
548                 if (inlinable(expr->fn)) {
549                         parse_inline(expr);
550                 }
551                 __pass_to_client(expr, CALL_HOOK_AFTER_INLINE);
552                 if (is_noreturn_func(expr->fn))
553                         nullify_path();
554                 handle_builtin_overflow_func(expr);
555                 break;
556         case EXPR_INITIALIZER:
557                 split_expr_list(expr->expr_list, expr);
558                 break;
559         case EXPR_IDENTIFIER:
560                 expr_set_parent_expr(expr->ident_expression, expr);
561                 __split_expr(expr->ident_expression);
562                 break;
563         case EXPR_INDEX:
564                 expr_set_parent_expr(expr->idx_expression, expr);
565                 __split_expr(expr->idx_expression);
566                 break;
567         case EXPR_POS:
568                 expr_set_parent_expr(expr->init_expr, expr);
569                 __split_expr(expr->init_expr);
570                 break;
571         case EXPR_SYMBOL:
572                 __pass_to_client(expr, SYM_HOOK);
573                 break;
574         case EXPR_STRING:
575                 __pass_to_client(expr, STRING_HOOK);
576                 break;
577         default:
578                 break;
579         };
580         pop_expression(&big_expression_stack);
581 }

583 static int is_forever_loop(struct statement *stmt)
584 {
585         struct expression *expr;
586         sval_t sval;

588         expr = strip_expr(stmt->iterator_pre_condition);
```



```
589         if (!expr)
590                 expr = stmt->iterator_post_condition;
591         if (!expr) {
592                 /* this is a for(;;) loop... */
593                 return 1;
594         }

596         if (get_value(expr, &sval) && sval.value != 0)
597                 return 1;

599         return 0;
600 }

602 static int loop_num;
603 static char *get_loop_name(int num)
604 {
605         char buf[256];

607         snprintf(buf, 255, "-loop%d", num);
608         buf[255] = '\0';
609         return alloc_sname(buf);
610 }

612 /*
613  * Pre Loops are while and for loops.
614  */
615 static void handle_pre_loop(struct statement *stmt)
616 {
617         int once_through; /* we go through the loop at least once */
618         struct sm_state *extra_sm = NULL;
619         int unchanged = 0;
620         char *loop_name;
621         struct stree *stree = NULL;
622         struct sm_state *sm = NULL;

624         loop_name = get_loop_name(loop_num);
625         loop_num++;

627         __split_stmt(stmt->iterator_pre_statement);
628         __prev_stmt = stmt->iterator_pre_statement;

630         once_through = implied_condition_true(stmt->iterator_pre_condition);

632         loop_count++;
633         __push_continues();
634         __push_breaks();

636         __merge_gotos(loop_name, NULL);

638         extra_sm = __extra_handle_canonical_loops(stmt, &stree);
639         __in_pre_condition++;
640         __pass_to_client(stmt, PRELOOP_HOOK);
641         __split_whole_condition(stmt->iterator_pre_condition);
642         __in_pre_condition--;
643         FOR_EACH_SM(stree, sm) {
644                 set_state(sm->owner, sm->name, sm->sym, sm->state);
645         } END_FOR_EACH_SM(sm);
646         free_stree(&stree);
647         if (extra_sm)
648                 extra_sm = get_sm_state(extra_sm->owner, extra_sm->name, extra_s

650         if (option_assume_loops)
651                 once_through = 1;

653         __split_stmt(stmt->iterator_statement);
654         if (is_forever_loop(stmt)) {
```

```
655                 __merge_continues();
656                 __save_gotos(loop_name, NULL);

658                 __push_fake_cur_stree();
659                 __split_stmt(stmt->iterator_post_statement);
660                 stree = __pop_fake_cur_stree();

662                 __discard_false_states();
663                 __use_breaks();

665                 if (!__path_is_null())
666                         __merge_stree_into_cur(stree);
667                 free_stree(&stree);
668         } else {
669                 __merge_continues();
670                 unchanged = __iterator_unchanged(extra_sm);
671                 __split_stmt(stmt->iterator_post_statement);
672                 __prev_stmt = stmt->iterator_post_statement;
673                 __cur_stmt = stmt;

675                 __save_gotos(loop_name, NULL);
676                 __in_pre_condition++;
677                 __split_whole_condition(stmt->iterator_pre_condition);
678                 __in_pre_condition--;
679                 nullify_path();
680                 __merge_false_states();
681                 if (once_through)
682                         __discard_false_states();
683                 else
684                         __merge_false_states();

686                 if (extra_sm && unchanged)
687                         __extra_pre_loop_hook_after(extra_sm,
688                                                     stmt->iterator_post_statement,
689                                                     stmt->iterator_pre_condition);
690                 __merge_breaks();
691         }
692         loop_count--;
693 }

695 /*
696  * Post loops are do {} while();
697  */
698 static void handle_post_loop(struct statement *stmt)
699 {
700         char *loop_name;

702         loop_name = get_loop_name(loop_num);
703         loop_num++;
704         loop_count++;

706         __push_continues();
707         __push_breaks();
708         __merge_gotos(loop_name, NULL);
709         __split_stmt(stmt->iterator_statement);
710         __merge_continues();
711         if (!is_zero(stmt->iterator_post_condition))
712                 __save_gotos(loop_name, NULL);

714         if (is_forever_loop(stmt)) {
715                 __use_breaks();
716         } else {
717                 __split_whole_condition(stmt->iterator_post_condition);
718                 __use_false_states();
719                 __merge_breaks();
720         }
```



```
721         loop_count--;
722 }

724 static int empty_statement(struct statement *stmt)
725 {
726         if (!stmt)
727                 return 0;
728         if (stmt->type == STMT_EXPRESSION && !stmt->expression)
729                 return 1;
730         return 0;
731 }

733 static int last_stmt_on_same_line(void)
734 {
735         struct statement *stmt;
736         int i = 0;

738         FOR_EACH_PTR_REVERSE(big_statement_stack, stmt) {
739                 if (!i++)
740                         continue;
741                 if  (stmt->pos.line == get_lineno())
742                         return 1;
743                 return 0;
744         } END_FOR_EACH_PTR_REVERSE(stmt);
745         return 0;
746 }

748 static void split_asm_constraints(struct expression_list *expr_list)
749 {
750         struct expression *expr;
751         int state = 0;

753         FOR_EACH_PTR(expr_list, expr) {
754                 switch (state) {
755                 case 0: /* identifier */
756                 case 1: /* constraint */
757                         state++;
758                         continue;
759                 case 2: /* expression */
760                         state = 0;
761                         __split_expr(expr);
762                         continue;
763                 }
764         } END_FOR_EACH_PTR(expr);
765 }

767 static int is_case_val(struct statement *stmt, sval_t sval)
768 {
769         sval_t case_sval;

771         if (stmt->type != STMT_CASE)
772                 return 0;
773         if (!stmt->case_expression) {
774                 __set_default();
775                 return 1;
776         }
777         if (!get_value(stmt->case_expression, &case_sval))
778                 return 0;
779         if (case_sval.value == sval.value)
780                 return 1;
781         return 0;
782 }

784 static struct range_list *get_case_rl(struct expression *switch_expr,
785                                       struct expression *case_expr,
786                                       struct expression *case_to)
```

```
787 {
788         sval_t start, end;
789         struct range_list *rl = NULL;
790         struct symbol *switch_type;
791
792         switch_type = get_type(switch_expr);
793         if (get_value(case_to, &end) && get_value(case_expr, &start)) {
794                 start = sval_cast(switch_type, start);
795                 end = sval_cast(switch_type, end);
796                 add_range(&rl, start, end);
797         } else if (get_value(case_expr, &start)) {
798                 start = sval_cast(switch_type, start);
799                 add_range(&rl, start, start);
800         }
801
802         return rl;
803 }
804
805 static void split_known_switch(struct statement *stmt, sval_t sval)
806 {
807         struct statement *tmp;
808         struct range_list *rl;
809
810         __split_expr(stmt->switch_expression);
811         sval = sval_cast(get_type(stmt->switch_expression), sval);
812
813         push_expression(&switch_expr_stack, stmt->switch_expression);
814         __save_switch_states(top_expression(switch_expr_stack));
815         nullify_path();
816         __push_default();
817         __push_breaks();
818
819         stmt = stmt->switch_statement;
820
821         __push_scope_hooks();
822         FOR_EACH_PTR(stmt->stmts, tmp) {
823                 __smatch_lineno = tmp->pos.line;
824                 if (is_case_val(tmp, sval)) {
825                         rl = alloc_rl(sval, sval);
826                         __merge_switches(top_expression(switch_expr_stack), rl);
827                         __pass_case_to_client(top_expression(switch_expr_stack),
828                 }
829                 if (__path_is_null())
830                         continue;
831                 __split_stmt(tmp);
832                 if (__path_is_null()) {
833                         __set_default();
834                         goto out;
835                 }
836         } END_FOR_EACH_PTR(tmp);
837 out:
838         __call_scope_hooks();
839         if (!__pop_default())
840                 __merge_switches(top_expression(switch_expr_stack), NULL);
841         __discard_switches();
842         __merge_breaks();
843         pop_expression(&switch_expr_stack);
844 }
845
846 static void split_case(struct statement *stmt)
847 {
848         struct range_list *rl = NULL;
849
850         expr_set_parent_stmt(stmt->case_expression, stmt);
851         expr_set_parent_stmt(stmt->case_to, stmt);
```



```
853         rl = get_case_rl(top_expression(switch_expr_stack),
854                          stmt->case_expression, stmt->case_to);
855         while (stmt->case_statement->type == STMT_CASE) {
856                 struct range_list *tmp;
857
858                 tmp = get_case_rl(top_expression(switch_expr_stack),
859                                   stmt->case_statement->case_expression,
860                                   stmt->case_statement->case_to);
861                 if (!tmp)
862                         break;
863                 rl = rl_union(rl, tmp);
864                 if (!stmt->case_expression)
865                         __set_default();
866                 stmt = stmt->case_statement;
867         }
868
869         __merge_switches(top_expression(switch_expr_stack), rl);
870
871         if (!stmt->case_expression)
872                 __set_default();
873         __split_stmt(stmt->case_statement);
874 }
875
876 int time_parsing_function(void)
877 {
878         return ms_since(&fn_start_time) / 1000;
879 }
880
881 static int taking_too_long(void)
882 {
883         if ((ms_since(&outer_fn_start_time) / 1000) > 60 * 5) /* five minutes */
884                 return 1;
885         return 0;
886 }
887
888 static int is_last_stmt(struct statement *cur_stmt)
889 {
890         struct symbol *fn;
891         struct statement *stmt;
892
893         if (!cur_func_sym)
894                 return 0;
895         fn = get_base_type(cur_func_sym);
896         if (!fn)
897                 return 0;
898         stmt = fn->stmt;
899         if (!stmt)
900                 stmt = fn->inline_stmt;
901         if (!stmt || stmt->type != STMT_COMPOUND)
902                 return 0;
903         stmt = last_ptr_list((struct ptr_list *)stmt->stmts);
904         if (stmt && stmt->type == STMT_LABEL)
905                 stmt = stmt->label_statement;
906         if (stmt == cur_stmt)
907                 return 1;
908         return 0;
909 }
910
911 static void handle_backward_goto(struct statement *goto_stmt)
912 {
913         const char *goto_name, *label_name;
914         struct statement *func_stmt;
915         struct symbol *base_type = get_base_type(cur_func_sym);
916         struct statement *tmp;
917         int found = 0;
```

```
919         if (!option_info)
920                 return;
921         if (last_goto_statement_handled)
922                 return;
923         last_goto_statement_handled = 1;

925         if (!goto_stmt->goto_label ||
926             goto_stmt->goto_label->type != SYM_LABEL ||
927             !goto_stmt->goto_label->ident)
928                 return;
929         goto_name = goto_stmt->goto_label->ident->name;

931         func_stmt = base_type->stmt;
932         if (!func_stmt)
933                 func_stmt = base_type->inline_stmt;
934         if (!func_stmt)
935                 return;
936         if (func_stmt->type != STMT_COMPOUND)
937                 return;

939         FOR_EACH_PTR(func_stmt->stmts, tmp) {
940                 if (!found) {
941                         if (tmp->type != STMT_LABEL)
942                                 continue;
943                         if (!tmp->label_identifier ||
944                             tmp->label_identifier->type != SYM_LABEL ||
945                             !tmp->label_identifier->ident)
946                                 continue;
947                         label_name = tmp->label_identifier->ident->name;
948                         if (strcmp(goto_name, label_name) != 0)
949                                 continue;
950                         found = 1;
951                 }
952                 __split_stmt(tmp);
953         } END_FOR_EACH_PTR(tmp);
954 }

956 static void fake_a_return(void)
957 {
958         struct symbol *return_type;

960         nullify_path();
961         __unnullify_path();

963         return_type = get_real_base_type(cur_func_sym);
964         return_type = get_real_base_type(return_type);
965         if (return_type != &void_ctype) {
966                 __pass_to_client(unknown_value_expression(NULL), RETURN_HOOK);
967                 nullify_path();
968         }
969 }

971 static void fake_an_empty_default(struct position pos)
972 {
973         static struct statement none = {};

975         none.pos = pos;
976         none.type = STMT_NONE;
977         __merge_switches(top_expression(switch_expr_stack), NULL);
978         __split_stmt(&none);
979 }

981 static void split_compound(struct statement *stmt)
982 {
983         struct statement *prev = NULL;
984         struct statement *cur = NULL;
```



```
985         struct statement *next;

987         __push_scope_hooks();

989         FOR_EACH_PTR(stmt->stmts, next) {
990                 /* just set them all ahead of time */
991                 stmt_set_parent_stmt(next, stmt);

993                 if (cur) {
994                         __prev_stmt = prev;
995                         __next_stmt = next;
996                         __cur_stmt = cur;
997                         __split_stmt(cur);
998                 }
999                 prev = cur;
1000                cur = next;
1001        } END_FOR_EACH_PTR(next);
1002        if (cur) {
1003                __prev_stmt = prev;
1004                __cur_stmt = cur;
1005                __next_stmt = NULL;
1006                __split_stmt(cur);
1007        }

1009        /*
1010         * For function scope, then delay calling the scope hooks until the
1011         * end of function hooks can run.  I'm not positive this is the right
1012         * thing...
1013         */
1014        if (!is_last_stmt(cur))
1015                __call_scope_hooks();
1016 }

1018 /*
1019  * This is a hack, work around for detecting empty functions.
1020  */
1021 static int need_delayed_scope_hooks(void)
1022 {
1023         struct symbol *fn = get_base_type(cur_func_sym);
1024         struct statement *stmt;

1026         if (!fn)
1027                 return 0;
1028         stmt = fn->stmt;
1029         if (!stmt)
1030                 stmt = fn->inline_stmt;
1031         if (stmt && stmt->type == STMT_COMPOUND)
1032                 return 1;
1033         return 0;
1034 }

1036 void __split_label_stmt(struct statement *stmt)
1037 {
1038         if (stmt->label_identifier &&
1039             stmt->label_identifier->type == SYM_LABEL &&
1040             stmt->label_identifier->ident) {
1041                 loop_count |= 0x0800000;
1042                 __merge_gotos(stmt->label_identifier->ident->name, stmt->label_i
1043         }
1044 }

1046 static void find_asm_gotos(struct statement *stmt)
1047 {
1048         struct symbol *sym;

1050         FOR_EACH_PTR(stmt->asm_labels, sym) {
```

```
1051                    __save_gotos(sym->ident->name, sym);
1052            } END_FOR_EACH_PTR(sym);
1053    }

1055    void __split_stmt(struct statement *stmt)
1056    {
1057            sval_t sval;

1059            if (!stmt)
1060                    goto out;

1062            if (!__in_fake_assign)
1063                    __silence_warnings_for_stmt = false;

1065            if (__bail_on_rest_of_function || is_skipped_function())
1066                    return;

1068            if (out_of_memory() || taking_too_long()) {
1069                    struct timeval stop;

1071                    gettimeofday(&stop, NULL);

1073                    __bail_on_rest_of_function = 1;
1074                    final_pass = 1;
1075                    sm_perror("Function too hairy.  Giving up. %lu seconds",
1076                              stop.tv_sec - fn_start_time.tv_sec);
1077                    fake_a_return();
1078                    final_pass = 0;  /* turn off sm_msg() from here */
1079                    return;
1080            }

1082            add_ptr_list(&big_statement_stack, stmt);
1083            free_expression_stack(&big_expression_stack);
1084            set_position(stmt->pos);
1085            __pass_to_client(stmt, STMT_HOOK);

1087            switch (stmt->type) {
1088            case STMT_DECLARATION:
1089                    split_declaration(stmt->declaration);
1090                    break;
1091            case STMT_RETURN:
1092                    expr_set_parent_stmt(stmt->ret_value, stmt);

1094                    __split_expr(stmt->ret_value);
1095                    __pass_to_client(stmt->ret_value, RETURN_HOOK);
1096                    __process_post_op_stack();
1097                    nullify_path();
1098                    break;
1099            case STMT_EXPRESSION:
1100                    expr_set_parent_stmt(stmt->expression, stmt);
1101                    expr_set_parent_stmt(stmt->context, stmt);

1103                    __split_expr(stmt->expression);
1104                    break;
1105            case STMT_COMPOUND:
1106                    split_compound(stmt);
1107                    break;
1108            case STMT_IF:
1109                    stmt_set_parent_stmt(stmt->if_true, stmt);
1110                    stmt_set_parent_stmt(stmt->if_false, stmt);
1111                    expr_set_parent_stmt(stmt->if_conditional, stmt);

1113                    if (known_condition_true(stmt->if_conditional)) {
1114                            __split_stmt(stmt->if_true);
1115                            break;
1116                    }
```



```
1117                    if (known_condition_false(stmt->if_conditional)) {
1118                            __split_stmt(stmt->if_false);
1119                            break;
1120                    }
1121                    __split_whole_condition(stmt->if_conditional);
1122                    __split_stmt(stmt->if_true);
1123                    if (empty_statement(stmt->if_true) &&
1124                            last_stmt_on_same_line() &&
1125                            !get_macro_name(stmt->if_true->pos))
1126                            sm_warning("if();");
1127                    __push_true_states();
1128                    __use_false_states();
1129                    __split_stmt(stmt->if_false);
1130                    __merge_true_states();
1131                    break;
1132            case STMT_ITERATOR:
1133                    stmt_set_parent_stmt(stmt->iterator_pre_statement, stmt);
1134                    stmt_set_parent_stmt(stmt->iterator_statement, stmt);
1135                    stmt_set_parent_stmt(stmt->iterator_post_statement, stmt);
1136                    expr_set_parent_stmt(stmt->iterator_pre_condition, stmt);
1137                    expr_set_parent_stmt(stmt->iterator_post_condition, stmt);

1139                    if (stmt->iterator_pre_condition)
1140                            handle_pre_loop(stmt);
1141                    else if (stmt->iterator_post_condition)
1142                            handle_post_loop(stmt);
1143                    else {
1144                            // these are for(;;) type loops.
1145                            handle_pre_loop(stmt);
1146                    }
1147                    break;
1148            case STMT_SWITCH:
1149                    stmt_set_parent_stmt(stmt->switch_statement, stmt);
1150                    expr_set_parent_stmt(stmt->switch_expression, stmt);

1152                    if (get_value(stmt->switch_expression, &sval)) {
1153                            split_known_switch(stmt, sval);
1154                            break;
1155                    }
1156                    __split_expr(stmt->switch_expression);
1157                    push_expression(&switch_expr_stack, stmt->switch_expression);
1158                    __save_switch_states(top_expression(switch_expr_stack));
1159                    nullify_path();
1160                    __push_default();
1161                    __push_breaks();
1162                    __split_stmt(stmt->switch_statement);
1163                    if (!__pop_default() && have_remaining_cases())
1164                            fake_an_empty_default(stmt->pos);
1165                    __discard_switches();
1166                    __merge_breaks();
1167                    pop_expression(&switch_expr_stack);
1168                    break;
1169            case STMT_CASE:
1170                    split_case(stmt);
1171                    break;
1172            case STMT_LABEL:
1173                    __split_label_stmt(stmt);
1174                    __split_stmt(stmt->label_statement);
1175                    break;
1176            case STMT_GOTO:
1177                    expr_set_parent_stmt(stmt->goto_expression, stmt);

1179                    __split_expr(stmt->goto_expression);
1180                    if (stmt->goto_label && stmt->goto_label->type == SYM_NODE) {
1181                            if (!strcmp(stmt->goto_label->ident->name, "break")) {
1182                                    __process_breaks();
```

```
1183                        } else if (!strcmp(stmt->goto_label->ident->name,
1184                                           "continue")) {
1185                                __process_continues();
1186                        }
1187                } else if (stmt->goto_label &&
1188                           stmt->goto_label->type == SYM_LABEL &&
1189                           stmt->goto_label->ident) {
1190                        __save_gotos(stmt->goto_label->ident->name, stmt->goto_l
1191                }
1192                nullify_path();
1193                if (is_last_stmt(stmt))
1194                        handle_backward_goto(stmt);
1195                break;
1196        case STMT_NONE:
1197                break;
1198        case STMT_ASM:
1199                expr_set_parent_stmt(stmt->asm_string, stmt);

1201                find_asm_gotos(stmt);
1202                __pass_to_client(stmt, ASM_HOOK);
1203                __split_expr(stmt->asm_string);
1204                split_asm_constraints(stmt->asm_outputs);
1205                split_asm_constraints(stmt->asm_inputs);
1206                split_asm_constraints(stmt->asm_clobbers);
1207                break;
1208        case STMT_CONTEXT:
1209                break;
1210        case STMT_RANGE:
1211                __split_expr(stmt->range_expression);
1212                __split_expr(stmt->range_low);
1213                __split_expr(stmt->range_high);
1214                break;
1215        }
1216        __pass_to_client(stmt, STMT_HOOK_AFTER);
1217 out:
1218        __process_post_op_stack();
1219 }

1221 static void split_expr_list(struct expression_list *expr_list, struct expression
1222 {
1223        struct expression *expr;

1225        FOR_EACH_PTR(expr_list, expr) {
1226                expr_set_parent_expr(expr, parent);
1227                __split_expr(expr);
1228                __process_post_op_stack();
1229        } END_FOR_EACH_PTR(expr);
1230 }

1232 static void split_sym(struct symbol *sym)
1233 {
1234        if (!sym)
1235                return;
1236        if (!(sym->namespace & NS_SYMBOL))
1237                return;

1239        __split_stmt(sym->stmt);
1240        __split_expr(sym->array_size);
1241        split_symlist(sym->arguments);
1242        split_symlist(sym->symbol_list);
1243        __split_stmt(sym->inline_stmt);
1244        split_symlist(sym->inline_symbol_list);
1245 }

1247 static void split_symlist(struct symbol_list *sym_list)
1248 {
```



```
1249        struct symbol *sym;

1251        FOR_EACH_PTR(sym_list, sym) {
1252                split_sym(sym);
1253        } END_FOR_EACH_PTR(sym);
1254 }

1256 typedef void (fake_cb)(struct expression *expr);

1258 static int member_to_number(struct expression *expr, struct ident *member)
1259 {
1260        struct symbol *type, *tmp;
1261        char *name;
1262        int i;

1264        if (!member)
1265                return -1;
1266        name = member->name;

1268        type = get_type(expr);
1269        if (!type || type->type != SYM_STRUCT)
1270                return -1;

1272        i = -1;
1273        FOR_EACH_PTR(type->symbol_list, tmp) {
1274                i++;
1275                if (!tmp->ident)
1276                        continue;
1277                if (strcmp(name, tmp->ident->name) == 0)
1278                        return i;
1279        } END_FOR_EACH_PTR(tmp);
1280        return -1;
1281 }

1283 static struct ident *number_to_member(struct expression *expr, int num)
1284 {
1285        struct symbol *type, *member;
1286        int i = 0;

1288        type = get_type(expr);
1289        if (!type || type->type != SYM_STRUCT)
1290                return NULL;

1292        FOR_EACH_PTR(type->symbol_list, member) {
1293                if (i == num)
1294                        return member->ident;
1295                i++;
1296        } END_FOR_EACH_PTR(member);
1297        return NULL;
1298 }

1300 static void fake_element_assigns_helper(struct expression *array, struct express

1302 static void set_inner_struct_members(struct expression *expr, struct symbol *mem
1303 {
1304        struct expression *edge_member, *assign;
1305        struct symbol *base = get_real_base_type(member);
1306        struct symbol *tmp;

1308        if (member->ident)
1309                expr = member_expression(expr, '.', member->ident);

1311        FOR_EACH_PTR(base->symbol_list, tmp) {
1312                struct symbol *type;

1314                type = get_real_base_type(tmp);
```

```
1315                 if (!type)
1316                         continue;
1317
1318                 edge_member = member_expression(expr, '.', tmp->ident);
1319                 if (get_extra_state(edge_member))
1320                         continue;
1321
1322                 if (type->type == SYM_UNION || type->type == SYM_STRUCT) {
1323                         set_inner_struct_members(expr, tmp);
1324                         continue;
1325                 }
1326
1327                 if (!tmp->ident)
1328                         continue;
1329
1330                 assign = assign_expression(edge_member, '=', zero_expr());
1331                 __split_expr(assign);
1332         } END_FOR_EACH_PTR(tmp);
1333
1334
1335 }
1336
1337 static void set_unset_to_zero(struct symbol *type, struct expression *expr)
1338 {
1339         struct symbol *tmp;
1340         struct expression *member = NULL;
1341         struct expression *assign;
1342         int op = '*';
1343
1344         if (expr->type == EXPR_PREOP && expr->op == '&') {
1345                 expr = strip_expr(expr->unop);
1346                 op = '.';
1347         }
1348
1349         FOR_EACH_PTR(type->symbol_list, tmp) {
1350                 type = get_real_base_type(tmp);
1351                 if (!type)
1352                         continue;
1353
1354                 if (tmp->ident) {
1355                         member = member_expression(expr, op, tmp->ident);
1356                         if (get_extra_state(member))
1357                                 continue;
1358                 }
1359
1360                 if (type->type == SYM_UNION || type->type == SYM_STRUCT) {
1361                         set_inner_struct_members(expr, tmp);
1362                         continue;
1363                 }
1364                 if (type->type == SYM_ARRAY)
1365                         continue;
1366                 if (!tmp->ident)
1367                         continue;
1368
1369                 assign = assign_expression(member, '=', zero_expr());
1370                 __split_expr(assign);
1371         } END_FOR_EACH_PTR(tmp);
1372 }
1373
1374 static void fake_member_assigns_helper(struct expression *symbol, struct express
1375 {
1376         struct expression *deref, *assign, *tmp, *right;
1377         struct symbol *struct_type, *type;
1378         struct ident *member;
1379         int member_idx;
```



```
1381         struct_type = get_type(symbol);
1382         if (!struct_type ||
1383             (struct_type->type != SYM_STRUCT && struct_type->type != SYM_UNION))
1384                 return;
1385
1386         /*
1387          * We're parsing an initializer that could look something like this:
1388          * struct foo foo = {
1389          *      42,
1390          *      .whatever.xxx = 11,
1391          *      .zzz = 12,
1392          * };
1393          *
1394          * So what we have here is a list with 42, .whatever, and .zzz.  We need
1395          * to break it up into left and right sides of the assignments.
1396          *
1397          */
1398         member_idx = 0;
1399         FOR_EACH_PTR(members, tmp) {
1400                 deref = NULL;
1401                 if (tmp->type == EXPR_IDENTIFIER) {
1402                         member_idx = member_to_number(symbol, tmp->expr_ident);
1403                         while (tmp->type == EXPR_IDENTIFIER) {
1404                                 member = tmp->expr_ident;
1405                                 tmp = tmp->ident_expression;
1406                                 if (deref)
1407                                         deref = member_expression(deref, '.', me
1408                                 else
1409                                         deref = member_expression(symbol, '.', m
1410                         }
1411                 } else {
1412                         member = number_to_member(symbol, member_idx);
1413                         deref = member_expression(symbol, '.', member);
1414                 }
1415                 right = tmp;
1416                 member_idx++;
1417                 if (right->type == EXPR_INITIALIZER) {
1418                         type = get_type(deref);
1419                         if (type && type->type == SYM_ARRAY)
1420                                 fake_element_assigns_helper(deref, right->expr_l
1421                         else
1422                                 fake_member_assigns_helper(deref, right->expr_li
1423                 } else {
1424                         assign = assign_expression(deref, '=', right);
1425                         fake_cb(assign);
1426                 }
1427         } END_FOR_EACH_PTR(tmp);
1428
1429         set_unset_to_zero(struct_type, symbol);
1430 }
1431
1432 static void fake_member_assigns(struct symbol *sym, fake_cb *fake_cb)
1433 {
1434         fake_member_assigns_helper(symbol_expression(sym),
1435                                    sym->initializer->expr_list, fake_cb);
1436 }
1437
1438 static void fake_element_assigns_helper(struct expression *array, struct express
1439 {
1440         struct expression *offset, *binop, *assign, *tmp;
1441         struct symbol *type;
1442         int idx;
1443
1444         if (ptr_list_size((struct ptr_list *)expr_list) > 1000)
1445                 return;
```

```
1447         idx = 0;
1448         FOR_EACH_PTR(expr_list, tmp) {
1449                 if (tmp->type == EXPR_INDEX) {
1450                         if (tmp->idx_from != tmp->idx_to)
1451                                 return;
1452                         idx = tmp->idx_from;
1453                         if (!tmp->idx_expression)
1454                                 goto next;
1455                         tmp = tmp->idx_expression;
1456                 }
1457                 offset = value_expr(idx);
1458                 binop = array_element_expression(array, offset);
1459                 if (tmp->type == EXPR_INITIALIZER) {
1460                         type = get_type(binop);
1461                         if (type && type->type == SYM_ARRAY)
1462                                 fake_element_assigns_helper(binop, tmp->expr_lis
1463                         else
1464                                 fake_member_assigns_helper(binop, tmp->expr_list
1465                 } else {
1466                         assign = assign_expression(binop, '=', tmp);
1467                         fake_cb(assign);
1468                 }
1469 next:
1470                 idx++;
1471         } END_FOR_EACH_PTR(tmp);
1472 }

1474 static void fake_element_assigns(struct symbol *sym, fake_cb *fake_cb)
1475 {
1476         fake_element_assigns_helper(symbol_expression(sym), sym->initializer->ex
1477 }

1479 static void fake_assign_expr(struct symbol *sym)
1480 {
1481         struct expression *assign, *symbol;

1483         symbol = symbol_expression(sym);
1484         assign = assign_expression(symbol, '=', sym->initializer);
1485         __split_expr(assign);
1486 }

1488 static void do_initializer_stuff(struct symbol *sym)
1489 {
1490         if (!sym->initializer)
1491                 return;

1493         if (sym->initializer->type == EXPR_INITIALIZER) {
1494                 if (get_real_base_type(sym)->type == SYM_ARRAY)
1495                         fake_element_assigns(sym, __split_expr);
1496                 else
1497                         fake_member_assigns(sym, __split_expr);
1498         } else {
1499                 fake_assign_expr(sym);
1500         }
1501 }

1503 static void split_declaration(struct symbol_list *sym_list)
1504 {
1505         struct symbol *sym;

1507         FOR_EACH_PTR(sym_list, sym) {
1508                 __pass_to_client(sym, DECLARATION_HOOK);
1509                 do_initializer_stuff(sym);
1510                 split_sym(sym);
1511         } END_FOR_EACH_PTR(sym);
1512 }
```



```
1514 static void call_global_assign_hooks(struct expression *assign)
1515 {
1516         __pass_to_client(assign, GLOBAL_ASSIGNMENT_HOOK);
1517 }

1519 static void fake_global_assign(struct symbol *sym)
1520 {
1521         struct expression *assign, *symbol;

1523         if (get_real_base_type(sym)->type == SYM_ARRAY) {
1524                 if (sym->initializer && sym->initializer->type == EXPR_INITIALIZ
1525                         fake_element_assigns(sym, call_global_assign_hooks);
1526                 } else if (sym->initializer) {
1527                         symbol = symbol_expression(sym);
1528                         assign = assign_expression(symbol, '=', sym->initializer
1529                         __pass_to_client(assign, GLOBAL_ASSIGNMENT_HOOK);
1530                 } else {
1531                         fake_element_assigns_helper(symbol_expression(sym), NULL
1532                 }
1533         } else if (get_real_base_type(sym)->type == SYM_STRUCT) {
1534                 if (sym->initializer && sym->initializer->type == EXPR_INITIALIZ
1535                         fake_member_assigns(sym, call_global_assign_hooks);
1536                 } else if (sym->initializer) {
1537                         symbol = symbol_expression(sym);
1538                         assign = assign_expression(symbol, '=', sym->initializer
1539                         __pass_to_client(assign, GLOBAL_ASSIGNMENT_HOOK);
1540                 } else {
1541                         fake_member_assigns_helper(symbol_expression(sym), NULL,
1542                 }
1543         } else {
1544                 symbol = symbol_expression(sym);
1545                 if (sym->initializer) {
1546                         assign = assign_expression(symbol, '=', sym->initializer
1547                         __split_expr(assign);
1548                 } else {
1549                         assign = assign_expression(symbol, '=', zero_expr());
1550                 }
1551                 __pass_to_client(assign, GLOBAL_ASSIGNMENT_HOOK);
1552         }
1553 }

1555 static void start_function_definition(struct symbol *sym)
1556 {
1557         __in_function_def = 1;
1558         __pass_to_client(sym, FUNC_DEF_HOOK);
1559         __in_function_def = 0;
1560         __pass_to_client(sym, AFTER_DEF_HOOK);

1562 }

1564 static void split_function(struct symbol *sym)
1565 {
1566         struct symbol *base_type = get_base_type(sym);
1567         struct timeval stop;

1569         if (!base_type->stmt && !base_type->inline_stmt)
1570                 return;

1572         gettimeofday(&outer_fn_start_time, NULL);
1573         gettimeofday(&fn_start_time, NULL);
1574         cur_func_sym = sym;
1575         if (sym->ident)
1576                 cur_func = sym->ident->name;
1577         set_position(sym->pos);
1578         loop_count = 0;
```

```
1579         last_goto_statement_handled = 0;
1580         sm_debug("new function:  %s\n", cur_func);
1581         __stree_id = 0;
1582         if (option_two_passes) {
1583                 __unnullify_path();
1584                 loop_num = 0;
1585                 final_pass = 0;
1586                 start_function_definition(sym);
1587                 __split_stmt(base_type->stmt);
1588                 __split_stmt(base_type->inline_stmt);
1589                 nullify_path();
1590         }
1591         __unnullify_path();
1592         loop_num = 0;
1593         final_pass = 1;
1594         start_function_definition(sym);
1595         __split_stmt(base_type->stmt);
1596         __split_stmt(base_type->inline_stmt);
1597         __pass_to_client(sym, END_FUNC_HOOK);
1598         if (need_delayed_scope_hooks())
1599                 __call_scope_hooks();
1600         __pass_to_client(sym, AFTER_FUNC_HOOK);

1602         clear_all_states();

1604         gettimeofday(&stop, NULL);
1605         if (option_time && stop.tv_sec - fn_start_time.tv_sec > 2) {
1606                 final_pass++;
1607                 sm_msg("func_time: %lu", stop.tv_sec - fn_start_time.tv_sec);
1608                 final_pass--;
1609         }
1610         cur_func_sym = NULL;
1611         cur_func = NULL;
1612         free_data_info_allocs();
1613         free_expression_stack(&switch_expr_stack);
1614         __free_ptr_list((struct ptr_list **)&big_statement_stack);
1615         __bail_on_rest_of_function = 0;
1616 }

1618 static void save_flow_state(void)
1619 {
1620         __add_ptr_list(&backup, INT_PTR(loop_num << 2), 0);
1621         __add_ptr_list(&backup, INT_PTR(loop_count << 2), 0);
1622         __add_ptr_list(&backup, INT_PTR(final_pass << 2), 0);

1624         __add_ptr_list(&backup, big_statement_stack, 0);
1625         __add_ptr_list(&backup, big_expression_stack, 0);
1626         __add_ptr_list(&backup, big_condition_stack, 0);
1627         __add_ptr_list(&backup, switch_expr_stack, 0);

1629         __add_ptr_list(&backup, cur_func_sym, 0);

1631         __add_ptr_list(&backup, __prev_stmt, 0);
1632         __add_ptr_list(&backup, __cur_stmt, 0);
1633         __add_ptr_list(&backup, __next_stmt, 0);

1635 }

1637 static void *pop_backup(void)
1638 {
1639         void *ret;

1641         ret = last_ptr_list(backup);
1642         delete_ptr_list_last(&backup);
1643         return ret;
1644 }
```



```
1646 static void restore_flow_state(void)
1647 {
1648         __next_stmt = pop_backup();
1649         __cur_stmt = pop_backup();
1650         __prev_stmt = pop_backup();

1652         cur_func_sym = pop_backup();
1653         switch_expr_stack = pop_backup();
1654         big_condition_stack = pop_backup();
1655         big_expression_stack = pop_backup();
1656         big_statement_stack = pop_backup();
1657         final_pass = PTR_INT(pop_backup()) >> 2;
1658         loop_count = PTR_INT(pop_backup()) >> 2;
1659         loop_num = PTR_INT(pop_backup()) >> 2;
1660 }

1662 static void parse_inline(struct expression *call)
1663 {
1664         struct symbol *base_type;
1665         char *cur_func_bak = cur_func;  /* not aligned correctly for backup */
1666         struct timeval time_backup = fn_start_time;
1667         struct expression *orig_inline = __inline_fn;
1668         int orig_budget;

1670         if (out_of_memory() || taking_too_long())
1671                 return;

1673         save_flow_state();

1675         __pass_to_client(call, INLINE_FN_START);
1676         final_pass = 0;  /* don't print anything */
1677         __inline_fn = call;
1678         orig_budget = inline_budget;
1679         inline_budget = inline_budget - 5;

1681         base_type = get_base_type(call->fn->symbol);
1682         cur_func_sym = call->fn->symbol;
1683         if (call->fn->symbol->ident)
1684                 cur_func = call->fn->symbol->ident->name;
1685         else
1686                 cur_func = NULL;
1687         set_position(call->fn->symbol->pos);

1689         save_all_states();
1690         big_statement_stack = NULL;
1691         big_expression_stack = NULL;
1692         big_condition_stack = NULL;
1693         switch_expr_stack = NULL;

1695         sm_debug("inline function:  %s\n", cur_func);
1696         __unnullify_path();
1697         loop_num = 0;
1698         loop_count = 0;
1699         start_function_definition(call->fn->symbol);
1700         __split_stmt(base_type->stmt);
1701         __split_stmt(base_type->inline_stmt);
1702         __pass_to_client(call->fn->symbol, END_FUNC_HOOK);
1703         __pass_to_client(call->fn->symbol, AFTER_FUNC_HOOK);

1705         free_expression_stack(&switch_expr_stack);
1706         __free_ptr_list((struct ptr_list **)&big_statement_stack);
1707         nullify_path();
1708         free_goto_stack();

1710         restore_flow_state();
```

```
1711         fn_start_time = time_backup;
1712         cur_func = cur_func_bak;

1714         restore_all_states();
1715         set_position(call->pos);
1716         __inline_fn = orig_inline;
1717         inline_budget = orig_budget;
1718         __pass_to_client(call, INLINE_FN_END);
1719 }

1721 static struct symbol_list *inlines_called;
1722 static void add_inline_function(struct symbol *sym)
1723 {
1724         static struct symbol_list *already_added;
1725         struct symbol *tmp;

1727         FOR_EACH_PTR(already_added, tmp) {
1728                 if (tmp == sym)
1729                         return;
1730         } END_FOR_EACH_PTR(tmp);

1732         add_ptr_list(&already_added, sym);
1733         add_ptr_list(&inlines_called, sym);
1734 }

1736 static void process_inlines(void)
1737 {
1738         struct symbol *tmp;

1740         FOR_EACH_PTR(inlines_called, tmp) {
1741                 split_function(tmp);
1742         } END_FOR_EACH_PTR(tmp);
1743         free_ptr_list(&inlines_called);
1744 }

1746 static struct symbol *get_last_scoped_symbol(struct symbol_list *big_list, int u
1747 {
1748         struct symbol *sym;

1750         FOR_EACH_PTR_REVERSE(big_list, sym) {
1751                 if (!sym->scope)
1752                         continue;
1753                 if (use_static && sym->ctype.modifiers & MOD_STATIC)
1754                         return sym;
1755                 if (!use_static && !(sym->ctype.modifiers & MOD_STATIC))
1756                         return sym;
1757         } END_FOR_EACH_PTR_REVERSE(sym);

1759         return NULL;
1760 }

1762 static bool interesting_function(struct symbol *sym)
1763 {
1764         static int prev_stream = -1;
1765         static bool prev_answer;
1766         const char *filename;
1767         int len;

1769         if (!(sym->ctype.modifiers & MOD_INLINE))
1770                 return true;

1772         if (sym->pos.stream == prev_stream)
1773                 return prev_answer;

1775         prev_stream = sym->pos.stream;
1776         prev_answer = false;
```



```
1778         filename = stream_name(sym->pos.stream);
1779         len = strlen(filename);
1780         if (len > 0 && filename[len - 1] == 'c')
1781                 prev_answer = true;
1782         return prev_answer;
1783 }

1785 static void split_inlines_in_scope(struct symbol *sym)
1786 {
1787         struct symbol *base;
1788         struct symbol_list *scope_list;
1789         int stream;

1791         scope_list = sym->scope->symbols;
1792         stream = sym->pos.stream;

1794         /* find the last static symbol in the file */
1795         FOR_EACH_PTR_REVERSE(scope_list, sym) {
1796                 if (sym->pos.stream != stream)
1797                         continue;
1798                 if (sym->type != SYM_NODE)
1799                         continue;
1800                 base = get_base_type(sym);
1801                 if (!base)
1802                         continue;
1803                 if (base->type != SYM_FN)
1804                         continue;
1805                 if (!base->inline_stmt)
1806                         continue;
1807                 if (!interesting_function(sym))
1808                         continue;
1809                 add_inline_function(sym);
1810         } END_FOR_EACH_PTR_REVERSE(sym);

1812         process_inlines();
1813 }

1815 static void split_inlines(struct symbol_list *sym_list)
1816 {
1817         struct symbol *sym;

1819         sym = get_last_scoped_symbol(sym_list, 0);
1820         if (sym)
1821                 split_inlines_in_scope(sym);
1822         sym = get_last_scoped_symbol(sym_list, 1);
1823         if (sym)
1824                 split_inlines_in_scope(sym);
1825 }

1827 static struct stree *clone_estates_perm(struct stree *orig)
1828 {
1829         struct stree *ret = NULL;
1830         struct sm_state *tmp;

1832         FOR_EACH_SM(orig, tmp) {
1833                 set_state_stree_perm(&ret, tmp->owner, tmp->name, tmp->sym, clon
1834         } END_FOR_EACH_SM(tmp);

1836         return ret;
1837 }

1839 struct position last_pos;
1840 static void split_c_file_functions(struct symbol_list *sym_list)
1841 {
1842         struct symbol *sym;
```

```
1844         __unnullify_path();
1845         FOR_EACH_PTR(sym_list, sym) {
1846                 set_position(sym->pos);
1847                 if (sym->type != SYM_NODE || get_base_type(sym)->type != SYM_FN)
1848                         __pass_to_client(sym, BASE_HOOK);
1849                 fake_global_assign(sym);
1850                 }
1851         } END_FOR_EACH_PTR(sym);
1852         global_states = clone_estates_perm(get_all_states_stree(SMATCH_EXTRA));
1853         nullify_path();
1855         FOR_EACH_PTR(sym_list, sym) {
1856                 set_position(sym->pos);
1857                 last_pos = sym->pos;
1858                 if (!interesting_function(sym))
1859                         continue;
1860                 if (sym->type == SYM_NODE && get_base_type(sym)->type == SYM_FN)
1861                         split_function(sym);
1862                         process_inlines();
1863                 }
1864                 last_pos = sym->pos;
1865         } END_FOR_EACH_PTR(sym);
1866         split_inlines(sym_list);
1867         __pass_to_client(sym_list, END_FILE_HOOK);
1868 }
1870 static int final_before_fake;
1871 void init_fake_env(void)
1872 {
1873         if (!in_fake_env)
1874                 final_before_fake = final_pass;
1875         in_fake_env++;
1876         __push_fake_cur_stree();
1877         final_pass = 0;
1878 }
1880 void end_fake_env(void)
1881 {
1882         __pop_fake_cur_stree();
1883         in_fake_env--;
1884         if (!in_fake_env)
1885                 final_pass = final_before_fake;
1886 }
1888 static void open_output_files(char *base_file)
1889 {
1890         char buf[256];
1892         snprintf(buf, sizeof(buf), "%s.smatch", base_file);
1893         sm_outfd = fopen(buf, "w");
1894         if (!sm_outfd)
1895                 sm_fatal("Cannot open %s", buf);
1897         if (!option_info)
1898                 return;
1900         snprintf(buf, sizeof(buf), "%s.smatch.sql", base_file);
1901         sql_outfd = fopen(buf, "w");
1902         if (!sql_outfd)
1903                 sm_fatal("Error:  Cannot open %s", buf);
1905         snprintf(buf, sizeof(buf), "%s.smatch.caller_info", base_file);
1906         caller_info_fd = fopen(buf, "w");
1907         if (!caller_info_fd)
1908                 sm_fatal("Error:  Cannot open %s", buf);
```



```
1909 }
1911 void smatch(int argc, char **argv)
1912 {
1913         struct string_list *filelist = NULL;
1914         struct symbol_list *sym_list;
1915         struct timeval stop, start;
1916         char *path;
1917         int len;
1919         gettimeofday(&start, NULL);
1921         sparse_initialize(argc, argv, &filelist);
1922         set_valid_ptr_max();
1923         alloc_valid_ptr_rl();
1924         FOR_EACH_PTR_NOTAG(filelist, base_file) {
1925                 path = getcwd(NULL, 0);
1926                 free(full_base_file);
1927                 if (path) {
1928                         len = strlen(path) + 1 + strlen(base_file) + 1;
1929                         full_base_file = malloc(len);
1930                         snprintf(full_base_file, len, "%s/%s", path, base_file);
1931                 } else {
1932                         full_base_file = alloc_string(base_file);
1933                 }
1934                 if (option_file_output)
1935                         open_output_files(base_file);
1936                 sym_list = sparse_keep_tokens(base_file);
1937                 split_c_file_functions(sym_list);
1938         } END_FOR_EACH_PTR_NOTAG(base_file);
1940         gettimeofday(&stop, NULL);
1942         set_position(last_pos);
1943         if (option_time)
1944                 sm_msg("time: %lu", stop.tv_sec - start.tv_sec);
1945         if (option_mem)
1946                 sm_msg("mem: %luKb", get_max_memory());
1947 }
```

```
**********************************************************
    4902 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_fn_arg_link.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2016 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * What we're trying to do here is record links between function pointers and
 * function data.  If you have foo->function(foo->data); that's very easy.  But
 * the problem is maybe when you pass the function and the data as parameters.
 *
 */

#include "smatch.h"
#include <ctype.h>

static int my_id;

static void save_in_fn_ptr_data_link_table(struct expression *fn, struct express
{
        struct symbol *fn_sym, *arg_sym;
        struct symbol *type;
        char *fn_name, *arg_name;
        int sym_len;
        char fn_buf[128];
        char arg_buf[128];

        fn_name = expr_to_var_sym(fn, &fn_sym);
        arg_name = expr_to_var_sym(arg, &arg_sym);
        if (!fn_sym || !fn_sym->ident || !arg_sym || !fn_name || !arg_name)
                goto free;
        if (fn_sym != arg_sym)
                goto free;

        sym_len = fn_sym->ident->len;

        /* This is ignoring
         * net/mac80211/driver-ops.h:482 drv_sta_remove() FN: local->ops->sta_re
         * but ideally the restriction can be removed later.
         */
        if (strncmp(fn_name, arg_name, sym_len) != 0)
                goto free;

        type = get_real_base_type(fn_sym);
        if (!type)
                goto free;
        if (type->type != SYM_PTR)
                goto free;
        type = get_real_base_type(type);
```



```
        if (!type || type->type != SYM_STRUCT || !type->ident)
                goto free;

        snprintf(fn_buf, sizeof(fn_buf), "(struct %s)%s", type->ident->name,
                 fn_name + sym_len);

        snprintf(arg_buf, sizeof(arg_buf), "(struct %s)%s", type->ident->name,
                 arg_name + sym_len);

        sql_insert_fn_ptr_data_link(fn_buf, arg_buf);
free:
        free_string(arg_name);
        free_string(fn_name);
}

static int print_calls_parameter(struct expression *call)
{
        struct expression *arg;
        int fn_param, arg_param;
        char buf[32];

        fn_param = get_param_num(call->fn);
        if (fn_param < 0)
                return 0;

        arg = get_argument_from_call_expr(call->args, 0);
        if (!arg)
                return 0;

        arg_param = get_param_num(arg);
        if (arg_param < 0)
                return 0;

        snprintf(buf, sizeof(buf), "%d", arg_param);
        sql_insert_return_implies(FN_ARG_LINK, fn_param, "$", buf);
        return 0;
}

static int print_call_is_linked(struct expression *call)
{
        struct expression *fn, *tmp;
        struct expression *arg;
        struct symbol *fn_sym;
        struct symbol *arg_sym = NULL;
        int i;

        fn = strip_expr(call->fn);
        tmp = get_assigned_expr(fn);
        if (tmp)
                fn = tmp;
        if (fn->type != EXPR_DEREF || !fn->member)
                return 0;

        fn_sym = expr_to_sym(fn);
        if (!fn_sym)
                return 0;

        i = -1;
        FOR_EACH_PTR(call->args, arg) {
                i++;
                tmp = get_assigned_expr(arg);
                if (tmp)
                        arg = tmp;
                arg_sym = expr_to_sym(arg);
                if (arg_sym == fn_sym) {
                        save_in_fn_ptr_data_link_table(fn, arg);
```

```
127                         return 1;
128                 }
129         } END_FOR_EACH_PTR(arg);

131         return 0;
132 }

134 static int is_recursive_call(struct expression *call)
135 {
136         if (call->fn->type != EXPR_SYMBOL)
137                 return 0;
138         if (call->fn->symbol == cur_func_sym)
139                 return 1;
140         return 0;
141 }

143 static void check_passes_fn_and_data(struct expression *call, struct expression
144 {
145         struct expression *arg;
146         struct expression *tmp;
147         struct symbol *fn_sym, *arg_sym;
148         struct symbol *type;
149         int data_nr;
150         int fn_param, arg_param;

152         if (is_recursive_call(call))
153                 return;

155         type = get_type(fn);
156         if (!type || type->type != SYM_PTR)
157                 return;
158         type = get_real_base_type(type);
159         if (!type || type->type != SYM_FN)
160                 return;
161         tmp = get_assigned_expr(fn);
162         if (tmp)
163                 fn = tmp;

165         if (!isdigit(value[0]))
166                 return;
167         data_nr = atoi(value);
168         arg = get_argument_from_call_expr(call->args, data_nr);
169         if (!arg)
170                 return;
171         tmp = get_assigned_expr(arg);
172         if (tmp)
173                 arg = tmp;

175         fn_param = get_param_num(fn);
176         arg_param = get_param_num(arg);
177         if (fn_param >= 0 && arg_param >= 0) {
178                 char buf[32];

180                 snprintf(buf, sizeof(buf), "%d", arg_param);
181                 sql_insert_return_implies(FN_ARG_LINK, fn_param, "$", buf);
182                 return;
183         }

185         fn_sym = expr_to_sym(fn);
186         if (!fn_sym)
187                 return;
188         arg_sym = expr_to_sym(arg);
189         if (arg_sym != fn_sym)
190                 return;
191         save_in_fn_ptr_data_link_table(fn, tmp);
192 }
```



```
194 static void match_call_info(struct expression *call)
195 {
196         if (print_calls_parameter(call))
197                 return;
198         if (print_call_is_linked(call))
199                 return;
200 }

202 void register_fn_arg_link(int id)
203 {
204         my_id = id;

206         if (!option_info)
207                 return;

209         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
210         select_return_implies_hook(FN_ARG_LINK, &check_passes_fn_and_data);
211 }
```

```
**********************************************************
    3866 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_function_hashtable.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "smatch.h"
#include "cwchash/hashtable.h"

static inline unsigned int djb2_hash(void *ky)
{
        char *str = (char *)ky;
        unsigned long hash = 5381;
        int c;

        while ((c = *str++))
                hash = ((hash << 5) + hash) + c; /* hash * 33 + c */

        return hash;
}

static inline int equalkeys(void *k1, void *k2)
{
        return !strcmp((char *)k1, (char *)k2);
}

#define DEFINE_FUNCTION_ADD_HOOK(_name, _item_type, _list_type) \
void add_##_name(struct hashtable *table, const char *look_for, _item_type *valu
{                                                               \
        _list_type *list;                                       \
        char *key;                                              \
                                                                \
        key = alloc_string(look_for);                           \
        list = search_##_name(table, key);                      \
        if (!list) {                                            \
                add_ptr_list(&list, value);                     \
        } else {                                                \
                remove_##_name(table, key);                     \
                add_ptr_list(&list, value);                     \
        }                                                       \
        insert_##_name(table, key, list);                       \
}

static inline struct hashtable *create_function_hashtable(int size)
{
        return create_hashtable(size, djb2_hash, equalkeys);
```



```
}

static inline void destroy_function_hashtable(struct hashtable *table)
{
        hashtable_destroy(table, 0);
}

#define DEFINE_FUNCTION_HASHTABLE(_name, _item_type, _list_type)   \
        DEFINE_HASHTABLE_INSERT(insert_##_name, char, _list_type); \
        DEFINE_HASHTABLE_SEARCH(search_##_name, char, _list_type); \
        DEFINE_HASHTABLE_REMOVE(remove_##_name, char, _list_type); \
        DEFINE_FUNCTION_ADD_HOOK(_name, _item_type, _list_type);

#define DEFINE_FUNCTION_HASHTABLE_STATIC(_name, _item_type, _list_type)   \
        static DEFINE_HASHTABLE_INSERT(insert_##_name, char, _list_type); \
        static DEFINE_HASHTABLE_SEARCH(search_##_name, char, _list_type); \
        static DEFINE_HASHTABLE_REMOVE(remove_##_name, char, _list_type); \
        static DEFINE_FUNCTION_ADD_HOOK(_name, _item_type, _list_type);

#define DEFINE_STRING_HASHTABLE_STATIC(_name)   \
        static DEFINE_HASHTABLE_INSERT(insert_##_name, char, int); \
        static DEFINE_HASHTABLE_SEARCH(search_##_name, char, int); \
        static struct hashtable *_name

static inline void load_hashtable_helper(const char *file, int (*insert_func)(st
{
        char filename[256];
        struct token *token;
        char *name;

        snprintf(filename, sizeof(filename), "%s.%s", option_project_str, file);
        token = get_tokens_file(filename);
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                name = alloc_string(show_ident(token->ident));
                insert_func(table, name, (void *)1);
                token = token->next;
        }
        clear_token_alloc();
}

#define load_strings(file, _table) load_hashtable_helper(file, insert_##_table,
```

```
**********************************************************
   31080 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_function_hooks.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * There are several types of function hooks:
 * add_function_hook()        - For any time a function is called.
 * add_function_assign_hook() - foo = the_function().
 * add_implied_return_hook()  - Calculates the implied return value.
 * add_macro_assign_hook()    - foo = the_macro().
 * return_implies_state()     - For when a return value of 1 implies locked
 *                              and 0 implies unlocked. etc. etc.
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"
#include "smatch_function_hashtable.h"

struct fcall_back {
        int type;
        struct data_range *range;
        union {
                func_hook *call_back;
                implication_hook *ranged;
                implied_return_hook *implied_return;
        } u;
        void *info;
};

ALLOCATOR(fcall_back, "call backs");
DECLARE_PTR_LIST(call_back_list, struct fcall_back);

DEFINE_FUNCTION_HASHTABLE_STATIC(callback, struct fcall_back, struct call_back_l
static struct hashtable *func_hash;

int __in_fake_parameter_assign;

#define REGULAR_CALL       0
#define RANGED_CALL        1
#define ASSIGN_CALL        2
#define IMPLIED_RETURN     3
#define MACRO_ASSIGN       4
#define MACRO_ASSIGN_EXTRA 5
```



```
struct return_implies_callback {
        int type;
        return_implies_hook *callback;
};
ALLOCATOR(return_implies_callback, "return_implies callbacks");
DECLARE_PTR_LIST(db_implies_list, struct return_implies_callback);
static struct db_implies_list *db_return_states_list;

typedef void (void_fn)(void);
DECLARE_PTR_LIST(void_fn_list, void_fn *);
static struct void_fn_list *return_states_before;
static struct void_fn_list *return_states_after;

static struct fcall_back *alloc_fcall_back(int type, void *call_back,
                                           void *info)
{
        struct fcall_back *cb;

        cb = __alloc_fcall_back(0);
        cb->type = type;
        cb->u.call_back = call_back;
        cb->info = info;
        return cb;
}

void add_function_hook(const char *look_for, func_hook *call_back, void *info)
{
        struct fcall_back *cb;

        cb = alloc_fcall_back(REGULAR_CALL, call_back, info);
        add_callback(func_hash, look_for, cb);
}

void add_function_assign_hook(const char *look_for, func_hook *call_back,
                              void *info)
{
        struct fcall_back *cb;

        cb = alloc_fcall_back(ASSIGN_CALL, call_back, info);
        add_callback(func_hash, look_for, cb);
}

void add_implied_return_hook(const char *look_for,
                             implied_return_hook *call_back,
                             void *info)
{
        struct fcall_back *cb;

        cb = alloc_fcall_back(IMPLIED_RETURN, call_back, info);
        add_callback(func_hash, look_for, cb);
}

void add_macro_assign_hook(const char *look_for, func_hook *call_back,
                           void *info)
{
        struct fcall_back *cb;

        cb = alloc_fcall_back(MACRO_ASSIGN, call_back, info);
        add_callback(func_hash, look_for, cb);
}

void add_macro_assign_hook_extra(const char *look_for, func_hook *call_back,
                           void *info)
{
        struct fcall_back *cb;
```

```
128         cb = alloc_fcall_back(MACRO_ASSIGN_EXTRA, call_back, info);
129         add_callback(func_hash, look_for, cb);
130 }

132 void return_implies_state(const char *look_for, long long start, long long end,
133                           implication_hook *call_back, void *info)
134 {
135         struct fcall_back *cb;

137         cb = alloc_fcall_back(RANGED_CALL, call_back, info);
138         cb->range = alloc_range_perm(ll_to_sval(start), ll_to_sval(end));
139         add_callback(func_hash, look_for, cb);
140 }

142 void select_return_states_hook(int type, return_implies_hook *callback)
143 {
144         struct return_implies_callback *cb = __alloc_return_implies_callback(0);

146         cb->type = type;
147         cb->callback = callback;
148         add_ptr_list(&db_return_states_list, cb);
149 }

151 void select_return_states_before(void_fn *fn)
152 {
153         void_fn **p = malloc(sizeof(void_fn *));
154         *p = fn;
155         add_ptr_list(&return_states_before, p);
156 }

158 void select_return_states_after(void_fn *fn)
159 {
160         void_fn **p = malloc(sizeof(void_fn *));
161         *p = fn;
162         add_ptr_list(&return_states_after, p);
163 }

165 static void call_return_states_before_hooks(void)
166 {
167         void_fn **fn;

169         FOR_EACH_PTR(return_states_before, fn) {
170                 (*fn)();
171         } END_FOR_EACH_PTR(fn);
172 }

174 static void call_return_states_after_hooks(struct expression *expr)
175 {
176         void_fn **fn;

178         FOR_EACH_PTR(return_states_after, fn) {
179                 (*fn)();
180         } END_FOR_EACH_PTR(fn);
181         __pass_to_client(expr, FUNCTION_CALL_HOOK_AFTER_DB);
182 }

184 static int call_call_backs(struct call_back_list *list, int type,
185                            const char *fn, struct expression *expr)
186 {
187         struct fcall_back *tmp;
188         int handled = 0;

190         FOR_EACH_PTR(list, tmp) {
191                 if (tmp->type == type) {
192                         (tmp->u.call_back)(fn, expr, tmp->info);
```



```
193                         handled = 1;
194                 }
195         } END_FOR_EACH_PTR(tmp);

197         return handled;
198 }

200 static void call_ranged_call_backs(struct call_back_list *list,
201                                    const char *fn, struct expression *call_expr,
202                                    struct expression *assign_expr)
203 {
204         struct fcall_back *tmp;

206         FOR_EACH_PTR(list, tmp) {
207                 (tmp->u.ranged)(fn, call_expr, assign_expr, tmp->info);
208         } END_FOR_EACH_PTR(tmp);
209 }

211 static struct call_back_list *get_same_ranged_call_backs(struct call_back_list *
212                                                          struct data_range *drange)
213 {
214         struct call_back_list *ret = NULL;
215         struct fcall_back *tmp;

217         FOR_EACH_PTR(list, tmp) {
218                 if (tmp->type != RANGED_CALL)
219                         continue;
220                 if (ranges_equiv(tmp->range, drange))
221                         add_ptr_list(&ret, tmp);
222         } END_FOR_EACH_PTR(tmp);
223         return ret;
224 }

226 static int in_list_exact_sval(struct range_list *list, struct data_range *drange
227 {
228         struct data_range *tmp;

230         FOR_EACH_PTR(list, tmp) {
231                 if (ranges_equiv(tmp, drange))
232                         return 1;
233         } END_FOR_EACH_PTR(tmp);
234         return 0;
235 }

237 static int assign_ranged_funcs(const char *fn, struct expression *expr,
238                                struct call_back_list *call_backs)
239 {
240         struct fcall_back *tmp;
241         struct sm_state *sm;
242         char *var_name;
243         struct symbol *sym;
244         struct smatch_state *estate;
245         struct stree *tmp_stree;
246         struct stree *final_states = NULL;
247         struct range_list *handled_ranges = NULL;
248         struct call_back_list *same_range_call_backs = NULL;
249         int handled = 0;

251         if (!call_backs)
252                 return 0;

254         var_name = expr_to_var_sym(expr->left, &sym);
255         if (!var_name || !sym)
256                 goto free;

258         FOR_EACH_PTR(call_backs, tmp) {
```

```
259                 if (tmp->type != RANGED_CALL)
260                         continue;
261
262                 if (in_list_exact_sval(handled_ranges, tmp->range))
263                         continue;
264                 __push_fake_cur_stree();
265                 tack_on(&handled_ranges, tmp->range);
266
267                 same_range_call_backs = get_same_ranged_call_backs(call_backs, t
268                 call_ranged_call_backs(same_range_call_backs, fn, expr->right, e
269                 __free_ptr_list((struct ptr_list **)&same_range_call_backs);
270
271                 estate = alloc_estate_range(tmp->range->min, tmp->range->max);
272                 set_extra_mod(var_name, sym, expr->left, estate);
273
274                 tmp_stree = __pop_fake_cur_stree();
275                 merge_fake_stree(&final_states, tmp_stree);
276                 free_stree(&tmp_stree);
277                 handled = 1;
278         } END_FOR_EACH_PTR(tmp);
279
280         FOR_EACH_SM(final_states, sm) {
281                 __set_sm(sm);
282         } END_FOR_EACH_SM(sm);
283
284         free_stree(&final_states);
285 free:
286         free_string(var_name);
287         return handled;
288 }
289
290 static void call_implies_callbacks(int comparison, struct expression *expr, sval
291 {
292         struct call_back_list *call_backs;
293         struct fcall_back *tmp;
294         const char *fn;
295         struct data_range *value_range;
296         struct stree *true_states = NULL;
297         struct stree *false_states = NULL;
298         struct stree *tmp_stree;
299
300         *implied_true = NULL;
301         *implied_false = NULL;
302         if (expr->fn->type != EXPR_SYMBOL || !expr->fn->symbol)
303                 return;
304         fn = expr->fn->symbol->ident->name;
305         call_backs = search_callback(func_hash, (char *)expr->fn->symbol->ident-
306         if (!call_backs)
307                 return;
308         value_range = alloc_range(sval, sval);
309
310         /* set true states */
311         __push_fake_cur_stree();
312         FOR_EACH_PTR(call_backs, tmp) {
313                 if (tmp->type != RANGED_CALL)
314                         continue;
315                 if (!true_comparison_range_LR(comparison, tmp->range, value_rang
316                         continue;
317                 (tmp->u.ranged)(fn, expr, NULL, tmp->info);
318         } END_FOR_EACH_PTR(tmp);
319         tmp_stree = __pop_fake_cur_stree();
320         merge_fake_stree(&true_states, tmp_stree);
321         free_stree(&tmp_stree);
322
323         /* set false states */
324         __push_fake_cur_stree();
```



```
325         FOR_EACH_PTR(call_backs, tmp) {
326                 if (tmp->type != RANGED_CALL)
327                         continue;
328                 if (!false_comparison_range_LR(comparison, tmp->range, value_ran
329                         continue;
330                 (tmp->u.ranged)(fn, expr, NULL, tmp->info);
331         } END_FOR_EACH_PTR(tmp);
332         tmp_stree = __pop_fake_cur_stree();
333         merge_fake_stree(&false_states, tmp_stree);
334         free_stree(&tmp_stree);
335
336         *implied_true = true_states;
337         *implied_false = false_states;
338 }
339
340 struct db_callback_info {
341         int true_side;
342         int comparison;
343         struct expression *expr;
344         struct range_list *rl;
345         int left;
346         struct stree *stree;
347         struct db_implies_list *callbacks;
348         int prev_return_id;
349         int cull;
350         int has_states;
351         char *ret_str;
352         struct smatch_state *ret_state;
353         struct expression *var_expr;
354         int handled;
355 };
356
357 static void store_return_state(struct db_callback_info *db_info, const char *ret
358 {
359         db_info->ret_str = alloc_sname(ret_str),
360         db_info->ret_state = state;
361 }
362
363 static bool fake_a_param_assignment(struct expression *expr, const char *return_
364 {
365         struct expression *arg, *left, *right, *fake_assign;
366         char *p;
367         int param;
368         char buf[256];
369         char *str;
370
371         if (expr->type != EXPR_ASSIGNMENT || expr->op != '=')
372                 return false;
373         left = expr->left;
374         right = expr->right;
375
376         while (right->type == EXPR_ASSIGNMENT)
377                 right = strip_expr(right->right);
378         if (!right || right->type != EXPR_CALL)
379                 return false;
380
381         p = strchr(return_str, '[');
382         if (!p)
383                 return false;
384
385         p++;
386         if (p[0] == '=' && p[1] == '=')
387                 p += 2;
388         if (p[0] != '$')
389                 return false;
```

```
391         snprintf(buf, sizeof(buf), "%s", p);

393         p = buf;
394         p += 1;
395         param = strtol(p, &p, 10);

397         p = strchr(p, ']');
398         if (!p || *p != ']')
399                 return false;
400         *p = '\0';

402         arg = get_argument_from_call_expr(right->args, param);
403         if (!arg)
404                 return false;
405         /*
406          * This is a sanity check to prevent side effects from evaluating stuff
407          * twice.
408          */
409         str = expr_to_chunk_sym_vsl(arg, NULL, NULL);
410         if (!str)
411                 return false;
412         free_string(str);

414         right = gen_expression_from_key(arg, buf);
415         if (!right)  /* Mostly fails for binops like [$0 + 4032] */
416                 return false;
417         fake_assign = assign_expression(left, '=', right);
418         __in_fake_parameter_assign++;
419         __split_expr(fake_assign);
420         __in_fake_parameter_assign--;
421         return true;
422 }

424 static void set_return_state(struct expression *expr, struct db_callback_info *d
425 {
426         struct smatch_state *state;

428         if (!db_info->ret_state)
429                 return;

431         state = alloc_estate_rl(cast_rl(get_type(expr), clone_rl(estate_rl(db_in
432         set_extra_expr_mod(expr, state);
433         db_info->ret_state = NULL;
434         fake_a_param_assignment(db_info->expr, db_info->ret_str);
435         db_info->ret_str = NULL;
436 }

438 static void handle_ret_equals_param(char *ret_string, struct range_list *rl, str
439 {
440         char *str;
441         long long param;
442         struct expression *arg;
443         struct range_list *orig;

445         str = strstr(ret_string, "==$");
446         if (!str)
447                 return;
448         str += 3;
449         param = strtoll(str, NULL, 10);
450         arg = get_argument_from_call_expr(call->args, param);
451         if (!arg)
452                 return;
453         get_absolute_rl(arg, &orig);
454         rl = rl_intersection(orig, rl);
455         if (!rl)
456                 return;
```



```
457         set_extra_expr_nomod(arg, alloc_estate_rl(rl));
458 }

460 static int impossible_limit(struct expression *expr, int param, char *key, char
461 {
462         struct expression *arg;
463         struct smatch_state *state;
464         struct range_list *passed;
465         struct range_list *limit;
466         struct symbol *compare_type;

468         while (expr->type == EXPR_ASSIGNMENT)
469                 expr = strip_expr(expr->right);
470         if (expr->type != EXPR_CALL)
471                 return 0;

473         arg = get_argument_from_call_expr(expr->args, param);
474         if (!arg)
475                 return 0;

477         if (strcmp(key, "$") == 0) {
478                 if (!get_implied_rl(arg, &passed))
479                         return 0;

481                 compare_type = get_arg_type(expr->fn, param);
482         } else {
483                 char *name;
484                 struct symbol *sym;

486                 name = get_variable_from_key(arg, key, &sym);
487                 if (!name || !sym)
488                         return 0;

490                 state = get_state(SMATCH_EXTRA, name, sym);
491                 if (!state) {
492                         free_string(name);
493                         return 0;
494                 }
495                 passed = estate_rl(state);
496                 if (!passed || is_whole_rl(passed)) {
497                         free_string(name);
498                         return 0;
499                 }

501                 compare_type = get_member_type_from_key(arg, key);
502         }

504         passed = cast_rl(compare_type, passed);
505         call_results_to_rl(expr, compare_type, value, &limit);
506         if (!limit || is_whole_rl(limit))
507                 return 0;
508         if (possibly_true_rl(passed, SPECIAL_EQUAL, limit))
509                 return 0;
510         if (option_debug || local_debug)
511                 sm_msg("impossible: %d '%s' limit '%s' == '%s'", param, key, sho
512         return 1;
513 }

515 static int is_impossible_data(int type, struct expression *expr, int param, char
516 {
517         if (type == PARAM_LIMIT && impossible_limit(expr, param, key, value))
518                 return 1;
519         if (type == COMPARE_LIMIT && param_compare_limit_is_impossible(expr, par
520                 if (local_debug)
521                         sm_msg("param_compare_limit_is_impossible: %d %s %s", pa
522                 return 1;
```

```
523         }
524         return 0;
525 }

527 static int func_type_mismatch(struct expression *expr, const char *value)
528 {
529         struct symbol *type;

531         /* This makes faking returns easier */
532         if (!value || value[0] == '\0')
533                 return 0;

535         while (expr->type == EXPR_ASSIGNMENT)
536                 expr = strip_expr(expr->right);

538         /*
539          * Short cut:  We only care about function pointers that are struct
540          * members.
541          *
542          */
543         if (expr->fn->type == EXPR_SYMBOL)
544                 return 0;

546         type = get_type(expr->fn);
547         if (!type)
548                 return 0;
549         if (type->type == SYM_PTR)
550                 type = get_real_base_type(type);

552         if (strcmp(type_to_str(type), value) == 0)
553                 return 0;

555         return 1;
556 }

558 static int db_compare_callback(void *_info, int argc, char **argv, char **azColN
559 {
560         struct db_callback_info *db_info = _info;
561         struct range_list *var_rl = db_info->rl;
562         struct range_list *ret_range;
563         int type, param;
564         char *key, *value;
565         struct return_implies_callback *tmp;
566         struct stree *stree;
567         int return_id;
568         int comparison;

570         if (argc != 6)
571                 return 0;

573         return_id = atoi(argv[0]);
574         type = atoi(argv[2]);
575         param = atoi(argv[3]);
576         key = argv[4];
577         value = argv[5];

579         db_info->has_states = 1;
580         if (db_info->prev_return_id != -1 && type == INTERNAL) {
581                 set_return_state(db_info->var_expr, db_info);
582                 stree = __pop_fake_cur_stree();

584                 if (!db_info->cull)
585                         merge_fake_stree(&db_info->stree, stree);
586                 free_stree(&stree);
587                 __push_fake_cur_stree();
588                 db_info->cull = 0;
```



```
589         }
590         db_info->prev_return_id = return_id;

592         if (type == INTERNAL && func_type_mismatch(db_info->expr, value))
593                 db_info->cull = 1;
594         if (db_info->cull)
595                 return 0;
596         if (type == CULL_PATH) {
597                 db_info->cull = 1;
598                 return 0;
599         }

601         if (is_impossible_data(type, db_info->expr, param, key, value)) {
602                 db_info->cull = 1;
603                 return 0;
604         }

606         call_results_to_rl(db_info->expr, get_type(strip_expr(db_info->expr)), a
607         ret_range = cast_rl(get_type(db_info->expr), ret_range);
608         if (!ret_range)
609                 ret_range = alloc_whole_rl(get_type(db_info->expr));

611         comparison = db_info->comparison;
612         if (db_info->left)
613                 comparison = flip_comparison(comparison);

615         if (db_info->true_side) {
616                 if (!possibly_true_rl(var_rl, comparison, ret_range))
617                         return 0;
618                 if (type == PARAM_LIMIT)
619                         param_limit_implications(db_info->expr, param, key, valu
620                 filter_by_comparison(&var_rl, comparison, ret_range);
621                 filter_by_comparison(&ret_range, flip_comparison(comparison), va
622         } else {
623                 if (!possibly_false_rl(var_rl, comparison, ret_range))
624                         return 0;
625                 if (type == PARAM_LIMIT)
626                         param_limit_implications(db_info->expr, param, key, valu
627                 filter_by_comparison(&var_rl, negate_comparison(comparison), ret
628                 filter_by_comparison(&ret_range, flip_comparison(negate_comparis
629         }

631         handle_ret_equals_param(argv[1], ret_range, db_info->expr);

633         if (type == INTERNAL) {
634                 set_state(-1, "unnull_path", NULL, &true_state);
635                 __add_return_comparison(strip_expr(db_info->expr), argv[1]);
636                 __add_return_to_param_mapping(db_info->expr, argv[1]);
637                 store_return_state(db_info, argv[1], alloc_estate_rl(clone_rl(va
638         }

640         FOR_EACH_PTR(db_info->callbacks, tmp) {
641                 if (tmp->type == type)
642                         tmp->callback(db_info->expr, param, key, value);
643         } END_FOR_EACH_PTR(tmp);

645         return 0;
646 }

648 static void compare_db_return_states_callbacks(struct expression *left, int comp
649 {
650         struct stree *orig_states;
651         struct stree *stree;
652         struct stree *true_states;
653         struct stree *false_states;
654         struct sm_state *sm;
```

```
655         struct db_callback_info db_info = {};
656         struct expression *var_expr;
657         struct expression *call_expr;
658         struct range_list *rl;
659         int call_on_left;

661         orig_states = clone_stree(__get_cur_stree());

663         /* legacy cruft.  need to fix call_implies_callbacks(). */
664         call_on_left = 1;
665         call_expr = left;
666         var_expr = right;
667         if (left->type != EXPR_CALL) {
668                 call_on_left = 0;
669                 call_expr = right;
670                 var_expr = left;
671         }

673         get_absolute_rl(var_expr, &rl);

675         db_info.comparison = comparison;
676         db_info.expr = call_expr;
677         db_info.rl = rl;
678         db_info.left = call_on_left;
679         db_info.callbacks = db_return_states_list;
680         db_info.var_expr = var_expr;

682         call_return_states_before_hooks();

684         db_info.true_side = 1;
685         db_info.stree = NULL;
686         db_info.prev_return_id = -1;
687         __push_fake_cur_stree();
688         sql_select_return_states("return_id, return, type, parameter, key, value
689                                  call_expr, db_compare_callback, &db_info);
690         set_return_state(db_info.var_expr, &db_info);
691         stree = __pop_fake_cur_stree();
692         if (!db_info.cull) {
693                 set_return_state(db_info.var_expr, &db_info);
694                 merge_fake_stree(&db_info.stree, stree);
695         }
696         free_stree(&stree);
697         true_states = db_info.stree;
698         if (!true_states && db_info.has_states) {
699                 __push_fake_cur_stree();
700                 set_path_impossible();
701                 true_states = __pop_fake_cur_stree();
702         }

704         nullify_path();
705         __unnullify_path();
706         FOR_EACH_SM(orig_states, sm) {
707                 __set_sm_cur_stree(sm);
708         } END_FOR_EACH_SM(sm);

710         db_info.true_side = 0;
711         db_info.stree = NULL;
712         db_info.prev_return_id = -1;
713         db_info.cull = 0;
714         __push_fake_cur_stree();
715         sql_select_return_states("return_id, return, type, parameter, key, value
716                                  db_compare_callback, &db_info);
717         stree = __pop_fake_cur_stree();
718         if (!db_info.cull) {
719                 set_return_state(db_info.var_expr, &db_info);
720                 merge_fake_stree(&db_info.stree, stree);
```



```
721         }
722         free_stree(&stree);
723         false_states = db_info.stree;
724         if (!false_states && db_info.has_states) {
725                 __push_fake_cur_stree();
726                 set_path_impossible();
727                 false_states = __pop_fake_cur_stree();
728         }

730         nullify_path();
731         __unnullify_path();
732         FOR_EACH_SM(orig_states, sm) {
733                 __set_sm_cur_stree(sm);
734         } END_FOR_EACH_SM(sm);

736         free_stree(&orig_states);

738         FOR_EACH_SM(true_states, sm) {
739                 __set_true_false_sm(sm, NULL);
740         } END_FOR_EACH_SM(sm);
741         FOR_EACH_SM(false_states, sm) {
742                 __set_true_false_sm(NULL, sm);
743         } END_FOR_EACH_SM(sm);

745         free_stree(&true_states);
746         free_stree(&false_states);

748         call_return_states_after_hooks(call_expr);

750         FOR_EACH_SM(implied_true, sm) {
751                 __set_true_false_sm(sm, NULL);
752         } END_FOR_EACH_SM(sm);
753         FOR_EACH_SM(implied_false, sm) {
754                 __set_true_false_sm(NULL, sm);
755         } END_FOR_EACH_SM(sm);
756 }

758 void function_comparison(struct expression *left, int comparison, struct express
759 {
760         struct expression *var_expr;
761         struct expression *call_expr;
762         struct stree *implied_true = NULL;
763         struct stree *implied_false = NULL;
764         struct range_list *rl;
765         sval_t sval;
766         int call_on_left;

768         if (unreachable())
769                 return;

771         /* legacy cruft.  need to fix call_implies_callbacks(). */
772         call_on_left = 1;
773         call_expr = left;
774         var_expr = right;
775         if (left->type != EXPR_CALL) {
776                 call_on_left = 0;
777                 call_expr = right;
778                 var_expr = left;
779         }

781         get_absolute_rl(var_expr, &rl);

783         if (rl_to_sval(rl, &sval))
784                 call_implies_callbacks(comparison, call_expr, sval, call_on_left

786         compare_db_return_states_callbacks(left, comparison, right, implied_true
```

```
787         free_stree(&implied_true);
788         free_stree(&implied_false);
789 }

791 static void call_ranged_return_hooks(struct db_callback_info *db_info)
792 {
793         struct call_back_list *call_backs;
794         struct expression *expr;
795         struct fcall_back *tmp;
796         char *fn;

798         expr = strip_expr(db_info->expr);
799         while (expr->type == EXPR_ASSIGNMENT)
800                 expr = strip_expr(expr->right);
801         if (expr->type != EXPR_CALL ||
802             expr->fn->type != EXPR_SYMBOL)
803                 return;

805         fn = expr->fn->symbol_name->name;

807         call_backs = search_callback(func_hash, fn);
808         FOR_EACH_PTR(call_backs, tmp) {
809                 struct range_list *range_rl = NULL;

811                 if (tmp->type != RANGED_CALL)
812                         continue;
813                 add_range(&range_rl, tmp->range->min, tmp->range->max);
814                 range_rl = cast_rl(estate_type(db_info->ret_state), range_rl);
815                 if (possibly_true_rl(range_rl, SPECIAL_EQUAL, estate_rl(db_info-
816                         if (!possibly_true_rl(rl_invert(range_rl), SPECIAL_EQUAL
817                                 (tmp->u.ranged)(fn, expr, db_info->expr, tmp->in
818                         else
819                                 db_info->handled = -1;
820                 }
821         } END_FOR_EACH_PTR(tmp);
822 }

824 static int db_assign_return_states_callback(void *_info, int argc, char **argv,
825 {
826         struct db_callback_info *db_info = _info;
827         struct range_list *ret_range;
828         int type, param;
829         char *key, *value;
830         struct return_implies_callback *tmp;
831         struct stree *stree;
832         int return_id;

834         if (argc != 6)
835                 return 0;

837         return_id = atoi(argv[0]);
838         type = atoi(argv[2]);
839         param = atoi(argv[3]);
840         key = argv[4];
841         value = argv[5];

843         if (db_info->prev_return_id != -1 && type == INTERNAL) {
844                 call_ranged_return_hooks(db_info);
845                 set_return_state(db_info->expr->left, db_info);
846                 stree = __pop_fake_cur_stree();
847                 if (!db_info->cull)
848                         merge_fake_stree(&db_info->stree, stree);
849                 free_stree(&stree);
850                 __push_fake_cur_stree();
851                 db_info->cull = 0;
852         }
```



```
853         db_info->prev_return_id = return_id;

855         if (type == INTERNAL && func_type_mismatch(db_info->expr, value))
856                 db_info->cull = 1;
857         if (db_info->cull)
858                 return 0;
859         if (type == CULL_PATH) {
860                 db_info->cull = 1;
861                 return 0;
862         }
863         if (is_impossible_data(type, db_info->expr, param, key, value)) {
864                 db_info->cull = 1;
865                 return 0;
866         }

868         if (type == PARAM_LIMIT)
869                 param_limit_implications(db_info->expr, param, key, value);

871         db_info->handled = 1;
872         call_results_to_rl(db_info->expr->right, get_type(strip_expr(db_info->ex
873         if (!ret_range)
874                 ret_range = alloc_whole_rl(get_type(strip_expr(db_info->expr->ri
875         ret_range = cast_rl(get_type(db_info->expr->right), ret_range);

877         if (type == INTERNAL) {
878                 set_state(-1, "unnull_path", NULL, &true_state);
879                 __add_return_comparison(strip_expr(db_info->expr->right), argv[1
880                 __add_comparison_info(db_info->expr->left, strip_expr(db_info->e
881                 __add_return_to_param_mapping(db_info->expr, argv[1]);
882                 store_return_state(db_info, argv[1], alloc_estate_rl(ret_range))
883         }

885         FOR_EACH_PTR(db_return_states_list, tmp) {
886                 if (tmp->type == type)
887                         tmp->callback(db_info->expr, param, key, value);
888         } END_FOR_EACH_PTR(tmp);

890         return 0;
891 }

893 static int db_return_states_assign(struct expression *expr)
894 {
895         struct expression *right;
896         struct sm_state *sm;
897         struct stree *stree;
898         struct db_callback_info db_info = {};

900         right = strip_expr(expr->right);

902         db_info.prev_return_id = -1;
903         db_info.expr = expr;
904         db_info.stree = NULL;
905         db_info.handled = 0;

907         call_return_states_before_hooks();

909         __push_fake_cur_stree();
910         sql_select_return_states("return_id, return, type, parameter, key, value
911                         right, db_assign_return_states_callback, &db_info);
912         if (option_debug) {
913                 sm_msg("%s return_id %d return_ranges %s",
914                         db_info.cull ? "culled" : "merging",
915                         db_info.prev_return_id,
916                         db_info.ret_state ? db_info.ret_state->name : "'<empty>'
917         }
918         if (db_info.handled)
```

```
919                 call_ranged_return_hooks(&db_info);
920         set_return_state(db_info.expr->left, &db_info);
921         stree = __pop_fake_cur_stree();
922         if (!db_info.cull)
923                 merge_fake_stree(&db_info.stree, stree);
924         free_stree(&stree);
925
926         if (!db_info.stree && db_info.cull) { /* this means we culled everything
927                 set_extra_expr_mod(expr->left, alloc_estate_whole(get_type(expr-
928                 set_path_impossible();
929         }
930         FOR_EACH_SM(db_info.stree, sm) {
931                 __set_sm(sm);
932         } END_FOR_EACH_SM(sm);
933
934         free_stree(&db_info.stree);
935         call_return_states_after_hooks(right);
936
937         return db_info.handled;
938 }
939
940 static int handle_implied_return(struct expression *expr)
941 {
942         struct range_list *rl;
943
944         if (!get_implied_return(expr->right, &rl))
945                 return 0;
946         rl = cast_rl(get_type(expr->left), rl);
947         set_extra_expr_mod(expr->left, alloc_estate_rl(rl));
948         return 1;
949 }
950
951 static void match_assign_call(struct expression *expr)
952 {
953         struct call_back_list *call_backs;
954         const char *fn;
955         struct expression *right;
956         int handled = 0;
957         struct range_list *rl;
958
959         if (expr->op != '=')
960                 return;
961
962         right = strip_expr(expr->right);
963         if (right->fn->type != EXPR_SYMBOL || !right->fn->symbol) {
964                 handled |= db_return_states_assign(expr);
965                 if (!handled)
966                         goto assigned_unknown;
967                 return;
968         }
969         if (is_fake_call(right)) {
970                 set_extra_expr_mod(expr->left, alloc_estate_whole(get_type(expr-
971                 return;
972         }
973
974         fn = right->fn->symbol->ident->name;
975         call_backs = search_callback(func_hash, (char *)fn);
976
977         /*
978          * The ordering here is sort of important.
979          * One example, of how this matters is that when we do:
980          *
981          *      len = strlen(str);
982          *
983          * That is handled by smatch_common_functions.c and smatch_strlen.c.
984          * They use implied_return and function_assign_hook respectively.
```



```
985          * We want to get the implied return first before we do the function
986          * assignment hook otherwise we end up writing the wrong thing for len
987          * in smatch_extra.c because we assume that it already holds the
988          * strlen() when we haven't set it yet.
989          */
990
991         if (db_return_states_assign(expr) == 1)
992                 handled = 1;
993         else
994                 handled = assign_ranged_funcs(fn, expr, call_backs);
995         handled |= handle_implied_return(expr);
996
997
998         call_call_backs(call_backs, ASSIGN_CALL, fn, expr);
999
1000        if (handled)
1001                return;
1002
1003 assigned_unknown:
1004        get_absolute_rl(expr->right, &rl);
1005        rl = cast_rl(get_type(expr->left), rl);
1006        set_extra_expr_mod(expr->left, alloc_estate_rl(rl));
1007 }
1008
1009 static int db_return_states_callback(void *_info, int argc, char **argv, char **
1010 {
1011        struct db_callback_info *db_info = _info;
1012        struct range_list *ret_range;
1013        int type, param;
1014        char *key, *value;
1015        struct return_implies_callback *tmp;
1016        struct stree *stree;
1017        int return_id;
1018        char buf[64];
1019
1020        if (argc != 6)
1021                return 0;
1022
1023        return_id = atoi(argv[0]);
1024        type = atoi(argv[2]);
1025        param = atoi(argv[3]);
1026        key = argv[4];
1027        value = argv[5];
1028
1029        if (db_info->prev_return_id != -1 && type == INTERNAL) {
1030                stree = __pop_fake_cur_stree();
1031                if (!db_info->cull)
1032                        merge_fake_stree(&db_info->stree, stree);
1033                free_stree(&stree);
1034                __push_fake_cur_stree();
1035                __unnullify_path();
1036                db_info->cull = 0;
1037        }
1038        db_info->prev_return_id = return_id;
1039
1040        if (type == INTERNAL && func_type_mismatch(db_info->expr, value))
1041                db_info->cull = 1;
1042        if (db_info->cull)
1043                return 0;
1044        if (type == CULL_PATH) {
1045                db_info->cull = 1;
1046                return 0;
1047        }
1048        if (is_impossible_data(type, db_info->expr, param, key, value)) {
1049                db_info->cull = 1;
1050                return 0;
```

```
1051         }
1052
1053         if (type == PARAM_LIMIT)
1054                 param_limit_implications(db_info->expr, param, key, value);
1055
1056         call_results_to_rl(db_info->expr, get_type(strip_expr(db_info->expr)), a
1057         ret_range = cast_rl(get_type(db_info->expr), ret_range);
1058
1059         if (type == INTERNAL) {
1060                 set_state(-1, "unnull_path", NULL, &true_state);
1061                 __add_return_comparison(strip_expr(db_info->expr), argv[1]);
1062                 __add_return_to_param_mapping(db_info->expr, argv[1]);
1063         }
1064
1065
1066         FOR_EACH_PTR(db_return_states_list, tmp) {
1067                 if (tmp->type == type)
1068                         tmp->callback(db_info->expr, param, key, value);
1069         } END_FOR_EACH_PTR(tmp);
1070
1071         /*
1072          * We want to store the return values so that we can split the strees
1073          * in smatch_db.c.  This uses set_state() directly because it's not a
1074          * real smatch_extra state.
1075          */
1076         snprintf(buf, sizeof(buf), "return %p", db_info->expr);
1077         set_state(SMATCH_EXTRA, buf, NULL, alloc_estate_rl(ret_range));
1078
1079         return 0;
1080 }
1081
1082 static void db_return_states(struct expression *expr)
1083 {
1084         struct sm_state *sm;
1085         struct stree *stree;
1086         struct db_callback_info db_info = {};
1087
1088         if (!__get_cur_stree())  /* no return functions */
1089                 return;
1090
1091         db_info.prev_return_id = -1;
1092         db_info.expr = expr;
1093         db_info.stree = NULL;
1094
1095         call_return_states_before_hooks();
1096
1097         __push_fake_cur_stree();
1098         __unnullify_path();
1099         sql_select_return_states("return_id, return, type, parameter, key, value
1100                         expr, db_return_states_callback, &db_info);
1101         stree = __pop_fake_cur_stree();
1102         if (!db_info.cull)
1103                 merge_fake_stree(&db_info.stree, stree);
1104         free_stree(&stree);
1105
1106         FOR_EACH_SM(db_info.stree, sm) {
1107                 __set_sm(sm);
1108         } END_FOR_EACH_SM(sm);
1109
1110         free_stree(&db_info.stree);
1111         call_return_states_after_hooks(expr);
1112 }
1113
1114 static int is_condition_call(struct expression *expr)
1115 {
1116         struct expression *tmp;
```



```
1117
1118         FOR_EACH_PTR_REVERSE(big_condition_stack, tmp) {
1119                 if (expr == tmp || expr_get_parent_expr(expr) == tmp)
1120                         return 1;
1121                 if (tmp->pos.line < expr->pos.line)
1122                         return 0;
1123         } END_FOR_EACH_PTR_REVERSE(tmp);
1124
1125         return 0;
1126 }
1127
1128 static void db_return_states_call(struct expression *expr)
1129 {
1130         if (unreachable())
1131                 return;
1132
1133         if (is_assigned_call(expr))
1134                 return;
1135         if (is_condition_call(expr))
1136                 return;
1137         db_return_states(expr);
1138 }
1139
1140 static void match_function_call(struct expression *expr)
1141 {
1142         struct call_back_list *call_backs;
1143
1144         if (expr->fn->type == EXPR_SYMBOL && expr->fn->symbol) {
1145                 call_backs = search_callback(func_hash, (char *)expr->fn->symbol
1146                 if (call_backs)
1147                         call_call_backs(call_backs, REGULAR_CALL,
1148                                         expr->fn->symbol->ident->name, expr);
1149         }
1150         db_return_states_call(expr);
1151 }
1152
1153 static void match_macro_assign(struct expression *expr)
1154 {
1155         struct call_back_list *call_backs;
1156         const char *macro;
1157         struct expression *right;
1158
1159         right = strip_expr(expr->right);
1160         macro = get_macro_name(right->pos);
1161         call_backs = search_callback(func_hash, (char *)macro);
1162         if (!call_backs)
1163                 return;
1164         call_call_backs(call_backs, MACRO_ASSIGN, macro, expr);
1165         call_call_backs(call_backs, MACRO_ASSIGN_EXTRA, macro, expr);
1166 }
1167
1168 int get_implied_return(struct expression *expr, struct range_list **rl)
1169 {
1170         struct call_back_list *call_backs;
1171         struct fcall_back *tmp;
1172         int handled = 0;
1173         char *fn;
1174
1175         *rl = NULL;
1176
1177         expr = strip_expr(expr);
1178         fn = expr_to_var(expr->fn);
1179         if (!fn)
1180                 goto out;
1181
1182         call_backs = search_callback(func_hash, fn);
```

```
1184         FOR_EACH_PTR(call_backs, tmp) {
1185                 if (tmp->type == IMPLIED_RETURN) {
1186                         (tmp->u.implied_return)(expr, tmp->info, rl);
1187                         handled = 1;
1188                 }
1189         } END_FOR_EACH_PTR(tmp);

1191 out:
1192         free_string(fn);
1193         return handled;
1194 }

1196 void create_function_hook_hash(void)
1197 {
1198         func_hash = create_function_hashtable(5000);
1199 }

1201 void register_function_hooks(int id)
1202 {
1203         add_hook(&match_function_call, CALL_HOOK_AFTER_INLINE);
1204         add_hook(&match_assign_call, CALL_ASSIGNMENT_HOOK);
1205         add_hook(&match_macro_assign, MACRO_ASSIGNMENT_HOOK);
1206 }
```

```
**********************************************************
    1142 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_function_info.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Record parameter types in the database.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static void match_def(struct symbol *sym)
{
        struct symbol *arg;
        int i;

        i = -1;
        FOR_EACH_PTR(sym->ctype.base_type->arguments, arg) {
                i++;
                sql_insert_function_type(i, type_to_str(get_real_base_type(arg))
        } END_FOR_EACH_PTR(arg);
}

void register_function_info(int id)
{
        my_id = id;
        add_hook(match_def, FUNC_DEF_HOOK);
}
```

```
**********************************************************
    7817 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_function_ptrs.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Track how functions are saved as various struct members or passed as
 * parameters.
 *
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static char *get_from__symbol_get(struct expression *expr)
{
        struct expression *arg;

        /*
         * typeof(&dib0070_attach) __a =
         * ((((typeof(&dib0070_attach)) (__symbol_get("dib0070_attach")))) ?:
         *  (__request_module(true, "symbol:" "dib0070_attach"), (((typeof(&dib0
         */

        expr = strip_expr(expr);

        if (expr->type != EXPR_CALL)
                return NULL;
        if (!sym_name_is("__symbol_get", expr->fn))
                return NULL;
        arg = get_argument_from_call_expr(expr->args, 0);
        if (!arg || arg->type != EXPR_STRING)
                return NULL;

        return alloc_string(arg->string->data);
}

static char *get_array_ptr(struct expression *expr)
{
        struct expression *array;
        struct symbol *type;
        char *name;
        char buf[256];

        array = get_array_base(expr);
```



```
        if (array) {
                name = get_member_name(array);
                if (name)
                        return name;
        }

        /* FIXME:  is_array() should probably be is_array_element() */
        type = get_type(expr);
        if (!array && type && type->type == SYM_ARRAY)
                array = expr;
        if (array) {
                name = expr_to_var(array);
                if (!name)
                        return NULL;
                snprintf(buf, sizeof(buf), "%s[]", name);
                return alloc_string(buf);
        }

        expr = get_assigned_expr(expr);
        array = get_array_base(expr);
        if (!array)
                return NULL;
        name = expr_to_var(array);
        if (!name)
                return NULL;
        snprintf(buf, sizeof(buf), "%s[]", name);
        free_string(name);
        return alloc_string(buf);
}

static int is_local_symbol(struct symbol *sym)
{
        if (!sym ||
            !(sym->ctype.modifiers & MOD_TOPLEVEL))
                return 1;
        return 0;
}

static char *ptr_prefix(struct symbol *sym)
{
        static char buf[128];


        if (is_local_symbol(sym))
                snprintf(buf, sizeof(buf), "%s ptr", get_function());
        else if (sym && toplevel(sym->scope))
                snprintf(buf, sizeof(buf), "%s ptr", get_base_file());
        else
                snprintf(buf, sizeof(buf), "ptr");

        return buf;
}

char *get_returned_ptr(struct expression *expr)
{
        struct symbol *type;
        char *name;
        char buf[256];

        if (expr->type != EXPR_CALL)
                return NULL;
        if (!expr->fn || expr->fn->type != EXPR_SYMBOL)
                return NULL;

        type = get_type(expr);
```

```
127         if (type && type->type == SYM_PTR)
128                 type = get_real_base_type(type);
129         if (!type || type->type != SYM_FN)
130                 return NULL;
131
132         name = expr_to_var(expr->fn);
133         if (!name)
134                 return NULL;
135         snprintf(buf, sizeof(buf), "r %s()", name);
136         free_string(name);
137         return alloc_string(buf);
138 }
139
140 char *get_fnptr_name(struct expression *expr)
141 {
142         char *name;
143
144         expr = strip_expr(expr);
145
146         /* (*ptrs[0])(a, b, c) is the same as ptrs[0](a, b, c); */
147         if (expr->type == EXPR_PREOP && expr->op == '*')
148                 expr = strip_expr(expr->unop);
149
150         name = get_from__symbol_get(expr);
151         if (name)
152                 return name;
153
154         name = get_array_ptr(expr);
155         if (name)
156                 return name;
157
158         name = get_returned_ptr(expr);
159         if (name)
160                 return name;
161
162         name = get_member_name(expr);
163         if (name)
164                 return name;
165
166         if (expr->type == EXPR_SYMBOL) {
167                 int param;
168                 char buf[256];
169                 struct symbol *sym;
170                 struct symbol *type;
171
172                 param = get_param_num_from_sym(expr->symbol);
173                 if (param >= 0) {
174                         snprintf(buf, sizeof(buf), "%s param %d", get_function()
175                         return alloc_string(buf);
176                 }
177
178                 name =  expr_to_var_sym(expr, &sym);
179                 if (!name)
180                         return NULL;
181                 type = get_type(expr);
182                 if (type && type->type == SYM_PTR) {
183                         snprintf(buf, sizeof(buf), "%s %s", ptr_prefix(sym), nam
184                         free_string(name);
185                         return alloc_string(buf);
186                 }
187                 return name;
188         }
189         return expr_to_var(expr);
190 }
191
192 static void match_passes_function_pointer(struct expression *expr)
```



```
193 {
194         struct expression *arg, *tmp;
195         struct symbol *type;
196         char *called_name;
197         char *fn_name;
198         char ptr_name[256];
199         int i;
200
201
202         i = -1;
203         FOR_EACH_PTR(expr->args, arg) {
204                 i++;
205
206                 tmp = strip_expr(arg);
207                 if (tmp->type == EXPR_PREOP && tmp->op == '&')
208                         tmp = strip_expr(tmp->unop);
209
210                 type = get_type(tmp);
211                 if (type && type->type == SYM_PTR)
212                         type = get_real_base_type(type);
213                 if (!type || type->type != SYM_FN)
214                         continue;
215
216                 called_name = expr_to_var(expr->fn);
217                 if (!called_name)
218                         return;
219                 fn_name = get_fnptr_name(tmp);
220                 if (!fn_name)
221                         goto free;
222
223                 snprintf(ptr_name, sizeof(ptr_name), "%s param %d", called_name,
224                 sql_insert_function_ptr(fn_name, ptr_name);
225 free:
226                 free_string(fn_name);
227                 free_string(called_name);
228         } END_FOR_EACH_PTR(arg);
229
230 }
231
232 static int get_row_count(void *_row_count, int argc, char **argv, char **azColNa
233 {
234         int *row_count = _row_count;
235
236         *row_count = 0;
237         if (argc != 1)
238                 return 0;
239         *row_count = atoi(argv[0]);
240         return 0;
241 }
242
243 static int can_hold_function_ptr(struct expression *expr)
244 {
245         struct symbol *type;
246
247         type = get_type(expr);
248         if (!type)
249                 return 0;
250         if (type->type == SYM_PTR || type->type == SYM_ARRAY) {
251                 type = get_real_base_type(type);
252                 if (!type)
253                         return 0;
254         }
255         if (type->type == SYM_FN)
256                 return 1;
257         if (type == &ulong_ctype && expr->type == EXPR_DEREF)
258                 return 1;
```

```
259         if (type == &void_ctype)
260                 return 1;
261         return 0;
262 }

264 static void match_function_assign(struct expression *expr)
265 {
266         struct expression *right;
267         struct symbol *type;
268         char *fn_name;
269         char *ptr_name;

271         if (__in_fake_assign)
272                 return;

274         right = strip_expr(expr->right);
275         if (right->type == EXPR_PREOP && right->op == '&')
276                 right = strip_expr(right->unop);

278         if (right->type != EXPR_SYMBOL &&
279             right->type != EXPR_DEREF)
280                 return;

282         if (!can_hold_function_ptr(right) ||
283             !can_hold_function_ptr(expr->left))
284                 return;

286         fn_name = get_fnptr_name(right);
287         ptr_name = get_fnptr_name(expr->left);
288         if (!fn_name || !ptr_name)
289                 goto free;
290         if (strcmp(fn_name, ptr_name) == 0)
291                 goto free;


294         type = get_type(right);
295         if (!type)
296                 return;
297         if (type->type == SYM_PTR || type->type == SYM_ARRAY) {
298                 type = get_real_base_type(type);
299                 if (!type)
300                         return;
301         }
302         if (type->type != SYM_FN) {
303                 int count = 0;

305                 /* look it up in function_ptr */
306                 run_sql(get_row_count, &count,
307                         "select count(*) from function_ptr where ptr = '%s'",
308                         fn_name);
309                 if (count == 0)
310                         goto free;
311         }

313         sql_insert_function_ptr(fn_name, ptr_name);
314 free:
315         free_string(fn_name);
316         free_string(ptr_name);
317 }

319 static void match_returns_function_pointer(struct expression *expr)
320 {
321         struct symbol *type;
322         char *fn_name;
323         char ptr_name[256];
```



```
325         if (__inline_fn)
326                 return;

328         type = get_real_base_type(cur_func_sym);
329         if (!type || type->type != SYM_FN)
330                 return;
331         type = get_real_base_type(type);
332         if (!type || type->type != SYM_PTR)
333                 return;
334         type = get_real_base_type(type);
335         if (!type || type->type != SYM_FN)
336                 return;

338         if (expr->type == EXPR_PREOP && expr->op == '&')
339                 expr = strip_expr(expr->unop);

341         fn_name = get_fnptr_name(expr);
342         if (!fn_name)
343                 return;
344         snprintf(ptr_name, sizeof(ptr_name), "r %s()", get_function());
345         sql_insert_function_ptr(fn_name, ptr_name);
346 }

348 void register_function_ptrs(int id)
349 {
350         my_id = id;

352         if (!option_info)
353                 return;

355         add_hook(&match_passes_function_pointer, FUNCTION_CALL_HOOK);
356         add_hook(&match_returns_function_pointer, RETURN_HOOK);
357         add_hook(&match_function_assign, ASSIGNMENT_HOOK);
358         add_hook(&match_function_assign, GLOBAL_ASSIGNMENT_HOOK);
359 }
```

```
**********************************************************
   23616 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_helper.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2006 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Miscellaneous helper functions.
 */

#include <stdlib.h>
#include <stdio.h>
#include "allocate.h"
#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

#define VAR_LEN 512

char *alloc_string(const char *str)
{
        char *tmp;

        if (!str)
                return NULL;
        tmp = malloc(strlen(str) + 1);
        strcpy(tmp, str);
        return tmp;
}

void free_string(char *str)
{
        free(str);
}

void remove_parens(char *str)
{
        char *src, *dst;

        dst = src = str;
        while (*src != '\0') {
                if (*src == '(' || *src == ')') {
                        src++;
                        continue;
                }
                *dst++ = *src++;
        }
        *dst = *src;
}
```



```
struct smatch_state *alloc_state_num(int num)
{
        struct smatch_state *state;
        static char buff[256];

        state = __alloc_smatch_state(0);
        snprintf(buff, 255, "%d", num);
        buff[255] = '\0';
        state->name = alloc_string(buff);
        state->data = INT_PTR(num);
        return state;
}

struct smatch_state *alloc_state_str(const char *name)
{
        struct smatch_state *state;

        state = __alloc_smatch_state(0);
        state->name = alloc_string(name);
        return state;
}

struct smatch_state *alloc_state_expr(struct expression *expr)
{
        struct smatch_state *state;
        char *name;

        state = __alloc_smatch_state(0);
        expr = strip_expr(expr);
        name = expr_to_str(expr);
        state->name = alloc_sname(name);
        free_string(name);
        state->data = expr;
        return state;
}

void append(char *dest, const char *data, int buff_len)
{
        strncat(dest, data, buff_len - strlen(dest) - 1);
}

/*
 * If you have "foo(a, b, 1);" then use
 * get_argument_from_call_expr(expr, 0) to return the expression for
 * a.  Yes, it does start counting from 0.
 */
struct expression *get_argument_from_call_expr(struct expression_list *args,
                                               int num)
{
        struct expression *expr;
        int i = 0;

        if (!args)
                return NULL;

        FOR_EACH_PTR(args, expr) {
                if (i == num)
                        return expr;
                i++;
        } END_FOR_EACH_PTR(expr);
        return NULL;
}

static struct expression *get_array_expr(struct expression *expr)
{
```

```
127         struct expression *parent;
128         struct symbol *type;

130         if (expr->type != EXPR_BINOP || expr->op != '+')
131                 return NULL;

133         type = get_type(expr->left);
134         if (!type)
135                 return NULL;
136         if (type->type == SYM_ARRAY)
137                 return expr->left;
138         if (type->type != SYM_PTR)
139                 return NULL;

141         parent = expr_get_parent_expr(expr);
142         if (!parent)  /* Sometimes we haven't set up the ->parent yet. FIXME!! *
143                 return expr->left;
144         if (parent->type == EXPR_PREOP && parent->op == '*')
145                 return expr->left;

147         return NULL;
148 }

150 static void __get_variable_from_expr(struct symbol **sym_ptr, char *buf,
151                                      struct expression *expr, int len,
152                                      int *complicated, int no_parens)
153 {


156         if (!expr) {
157                 /* can't happen on valid code */
158                 *complicated = 1;
159                 return;
160         }

162         switch (expr->type) {
163         case EXPR_DEREF: {
164                 struct expression *deref;
165                 int op;

167                 deref = expr->deref;
168                 op = deref->op;
169                 if (op == '*') {
170                         struct expression *unop = strip_expr(deref->unop);

172                         if (unop->type == EXPR_PREOP && unop->op == '&') {
173                                 deref = unop->unop;
174                                 op = '.';
175                         } else {
176                                 deref = deref->unop;
177                                 if (!is_pointer(deref))
178                                         op = '.';
179                         }
180                 }

182                 __get_variable_from_expr(sym_ptr, buf, deref, len, complicated,

184                 if (op == '*')
185                         append(buf, "->", len);
186                 else
187                         append(buf, ".", len);

189                 if (expr->member)
190                         append(buf, expr->member->name, len);
191                 else
192                         append(buf, "unknown_member", len);
```



```
194                 return;
195         }
196         case EXPR_SYMBOL:
197                 if (expr->symbol_name)
198                         append(buf, expr->symbol_name->name, len);
199                 if (sym_ptr) {
200                         if (*sym_ptr)
201                                 *complicated = 1;
202                         *sym_ptr = expr->symbol;
203                 }
204                 return;
205         case EXPR_PREOP: {
206                 const char *tmp;

208                 if (get_expression_statement(expr)) {
209                         *complicated = 2;
210                         return;
211                 }

213                 if (expr->op == '(') {
214                         if (!no_parens && expr->unop->type != EXPR_SYMBOL)
215                                 append(buf, "(", len);
216                 } else if (expr->op != '*' || !get_array_expr(expr->unop)) {
217                         tmp = show_special(expr->op);
218                         append(buf, tmp, len);
219                 }
220                 __get_variable_from_expr(sym_ptr, buf, expr->unop,
221                                                 len, complicated, no_parens);

223                 if (expr->op == '(' && !no_parens && expr->unop->type != EXPR_SY
224                         append(buf, ")", len);

226                 if (expr->op == SPECIAL_DECREMENT ||
227                                 expr->op == SPECIAL_INCREMENT)
228                         *complicated = 1;

230                 return;
231         }
232         case EXPR_POSTOP: {
233                 const char *tmp;

235                 __get_variable_from_expr(sym_ptr, buf, expr->unop,
236                                                 len, complicated, no_parens);
237                 tmp = show_special(expr->op);
238                 append(buf, tmp, len);

240                 if (expr->op == SPECIAL_DECREMENT || expr->op == SPECIAL_INCREME
241                         *complicated = 1;
242                 return;
243         }
244         case EXPR_ASSIGNMENT:
245         case EXPR_COMPARE:
246         case EXPR_LOGICAL:
247         case EXPR_BINOP: {
248                 char tmp[10];
249                 struct expression *array_expr;

251                 *complicated = 1;
252                 array_expr = get_array_expr(expr);
253                 if (array_expr) {
254                         __get_variable_from_expr(sym_ptr, buf, array_expr, len,
255                         append(buf, "[", len);
256                 } else {
257                         __get_variable_from_expr(sym_ptr, buf, expr->left, len,
258                         snprintf(tmp, sizeof(tmp), " %s ", show_special(expr->op
```

```
259                     append(buf, tmp, len);
260             }
261             __get_variable_from_expr(NULL, buf, expr->right, len, complicate
262             if (array_expr)
263                     append(buf, "]", len);
264             return;
265     }
266     case EXPR_VALUE: {
267             char tmp[25];
268
269             *complicated = 1;
270             snprintf(tmp, 25, "%lld", expr->value);
271             append(buf, tmp, len);
272             return;
273     }
274     case EXPR_STRING:
275             append(buf, "\"", len);
276             if (expr->string)
277                     append(buf, expr->string->data, len);
278             append(buf, "\"", len);
279             return;
280     case EXPR_CALL: {
281             struct expression *tmp;
282             int i;
283
284             *complicated = 1;
285             __get_variable_from_expr(NULL, buf, expr->fn, len, complicated,
286             append(buf, "(", len);
287             i = 0;
288             FOR_EACH_PTR(expr->args, tmp) {
289                     if (i++)
290                             append(buf, ", ", len);
291                     __get_variable_from_expr(NULL, buf, tmp, len, complicate
292             } END_FOR_EACH_PTR(tmp);
293             append(buf, ")", len);
294             return;
295     }
296     case EXPR_CAST:
297     case EXPR_FORCE_CAST:
298             __get_variable_from_expr(sym_ptr, buf,
299                                      expr->cast_expression, len,
300                                      complicated, no_parens);
301             return;
302     case EXPR_SIZEOF: {
303             sval_t sval;
304             int size;
305             char tmp[25];
306
307             if (expr->cast_type && get_base_type(expr->cast_type)) {
308                     size = type_bytes(get_base_type(expr->cast_type));
309                     snprintf(tmp, 25, "%d", size);
310                     append(buf, tmp, len);
311             } else if (get_value(expr, &sval)) {
312                     snprintf(tmp, 25, "%s", sval_to_str(sval));
313                     append(buf, tmp, len);
314             }
315             return;
316     }
317     case EXPR_IDENTIFIER:
318             *complicated = 1;
319             if (expr->expr_ident)
320                     append(buf, expr->expr_ident->name, len);
321             return;
322     default:
323             *complicated = 1;
324             //printf("unknown type = %d\n", expr->type);
```



```
325                 return;
326         }
327 }

329 struct expr_str_cache_results {
330         struct expression *expr;
331         int no_parens;
332         char str[VAR_LEN];
333         struct symbol *sym;
334         int complicated;
335 };

337 static void get_variable_from_expr(struct symbol **sym_ptr, char *buf,
338                                    struct expression *expr, int len,
339                                    int *complicated, int no_parens)
340 {
341         static struct expr_str_cache_results cached[8];
342         struct symbol *tmp_sym = NULL;
343         static int idx;
344         int i;

346         for (i = 0; i < ARRAY_SIZE(cached); i++) {
347                 if (expr == cached[i].expr &&
348                     no_parens == cached[i].no_parens) {
349                         strncpy(buf, cached[i].str, len);
350                         if (sym_ptr)
351                                 *sym_ptr = cached[i].sym;
352                         *complicated = cached[i].complicated;
353                         return;
354                 }
355         }

357         __get_variable_from_expr(&tmp_sym, buf, expr, len, complicated, no_paren
358         if (sym_ptr)
359                 *sym_ptr = tmp_sym;

361         cached[idx].expr = expr;
362         cached[idx].no_parens = no_parens;
363         strncpy(cached[idx].str, buf, VAR_LEN);
364         cached[idx].sym = tmp_sym;
365         cached[idx].complicated = *complicated;

367         idx = (idx + 1) % ARRAY_SIZE(cached);
368 }

370 /*
371  * This is returns a stylized "c looking" representation of the
372  * variable name.
373  *
374  * It uses the same buffer every time so you have to save the result
375  * yourself if you want to keep it.
376  *
377  */

379 char *expr_to_str_sym(struct expression *expr, struct symbol **sym_ptr)
380 {
381         static char var_name[VAR_LEN];
382         int complicated = 0;

384         if (sym_ptr)
385                 *sym_ptr = NULL;
386         var_name[0] = '\0';

388         if (!expr)
389                 return NULL;
390         get_variable_from_expr(sym_ptr, var_name, expr, sizeof(var_name),
```

```
391                                     &complicated, 0);
392         if (complicated < 2)
393                 return alloc_string(var_name);
394         else
395                 return NULL;
396 }

398 char *expr_to_str(struct expression *expr)
399 {
400         return expr_to_str_sym(expr, NULL);
401 }

403 /*
404  * get_variable_from_expr_simple() only returns simple variables.
405  * If it's a complicated variable like a->foo[x] instead of just 'a->foo'
406  * then it returns NULL.
407  */
408 char *expr_to_var_sym(struct expression *expr,
409                                         struct symbol **sym_ptr)
410 {
411         static char var_name[VAR_LEN];
412         int complicated = 0;

414         if (sym_ptr)
415                 *sym_ptr = NULL;
416         var_name[0] = '\0';

418         if (!expr)
419                 return NULL;
420         expr = strip_expr(expr);
421         get_variable_from_expr(sym_ptr, var_name, expr, sizeof(var_name),
422                                &complicated, 1);

424         if (complicated) {
425                 if (sym_ptr)
426                         *sym_ptr = NULL;
427                 return NULL;
428         }
429         return alloc_string(var_name);
430 }

432 char *expr_to_var(struct expression *expr)
433 {
434         return expr_to_var_sym(expr, NULL);
435 }

437 struct symbol *expr_to_sym(struct expression *expr)
438 {
439         struct symbol *sym;
440         char *name;

442         name = expr_to_var_sym(expr, &sym);
443         free_string(name);
444         return sym;
445 }

447 int get_complication_score(struct expression *expr)
448 {
449         expr = strip_expr(expr);

451         /*
452          * Don't forget to keep get_complication_score() and store_all_links()
453          * in sync.
454          *
455          */
```



```
457         if (!expr)
458                 return 990;

460         switch (expr->type) {
461         case EXPR_CALL:
462                 return 991;
463         case EXPR_COMPARE:
464         case EXPR_BINOP:
465                 return get_complication_score(expr->left) +
466                        get_complication_score(expr->right);
467         case EXPR_SYMBOL:
468                 return 1;
469         case EXPR_PREOP:
470                 if (expr->op == '*' || expr->op == '(')
471                         return get_complication_score(expr->unop);
472                 return 993;
473         case EXPR_DEREF:
474                 return get_complication_score(expr->deref);
475         case EXPR_VALUE:
476         case EXPR_SIZEOF:
477                 return 0;
478         default:
479                 return 994;
480         }
481 }

483 struct expression *reorder_expr_alphabetically(struct expression *expr)
484 {
485         struct expression *ret;
486         char *left, *right;

488         if (expr->type != EXPR_BINOP)
489                 return expr;
490         if (expr->op != '+' && expr->op != '*')
491                 return expr;

493         left = expr_to_var(expr->left);
494         right = expr_to_var(expr->right);
495         ret = expr;
496         if (!left || !right)
497                 goto free;
498         if (strcmp(left, right) <= 0)
499                 goto free;

501         ret = binop_expression(expr->right, expr->op, expr->left);
502 free:
503         free_string(left);
504         free_string(right);

506         return ret;
507 }

509 char *expr_to_chunk_helper(struct expression *expr, struct symbol **sym, struct
510 {
511         struct var_sym_list *tmp_vsl;
512         char *name;
513         struct symbol *tmp;
514         int score;

516         if (vsl)
517                 *vsl = NULL;
518         if (sym)
519                 *sym = NULL;

521         expr = strip_parens(expr);
522         if (!expr)
```

```
523                 return NULL;

525         name = expr_to_var_sym(expr, &tmp);
526         if (name && tmp) {
527                 if (sym)
528                         *sym = tmp;
529                 if (vsl)
530                         *vsl = expr_to_vsl(expr);
531                 return name;
532         }
533         free_string(name);

535         score = get_complication_score(expr);
536         if (score <= 0 || score > 2)
537                 return NULL;

539         tmp_vsl = expr_to_vsl(expr);
540         if (vsl) {
541                 *vsl = tmp_vsl;
542                 if (!*vsl)
543                         return NULL;
544         }
545         if (sym) {
546                 if (ptr_list_size((struct ptr_list *)tmp_vsl) == 1) {
547                         struct var_sym *vs;

549                         vs = first_ptr_list((struct ptr_list *)tmp_vsl);
550                         *sym = vs->sym;
551                 }
552         }

554         expr = reorder_expr_alphabetically(expr);

556         return expr_to_str(expr);
557 }

559 char *expr_to_known_chunk_sym(struct expression *expr, struct symbol **sym)
560 {
561         return expr_to_chunk_helper(expr, sym, NULL);
562 }

564 char *expr_to_chunk_sym_vsl(struct expression *expr, struct symbol **sym, struct
565 {
566         return expr_to_chunk_helper(expr, sym, vsl);
567 }

569 int sym_name_is(const char *name, struct expression *expr)
570 {
571         if (!expr)
572                 return 0;
573         if (expr->type != EXPR_SYMBOL)
574                 return 0;
575         if (!strcmp(expr->symbol_name->name, name))
576                 return 1;
577         return 0;
578 }

580 int is_zero(struct expression *expr)
581 {
582         sval_t sval;

584         if (get_value(expr, &sval) && sval.value == 0)
585                 return 1;
586         return 0;
587 }
```



```
589 int is_array(struct expression *expr)
590 {
591         struct symbol *type;

593         expr = strip_expr(expr);
594         if (!expr)
595                 return 0;

597         if (expr->type == EXPR_PREOP && expr->op == '*') {
598                 expr = strip_expr(expr->unop);
599                 if (!expr)
600                         return 0;
601                 if (expr->type == EXPR_BINOP && expr->op == '+')
602                         return 1;
603         }

605         if (expr->type != EXPR_BINOP || expr->op != '+')
606                 return 0;

608         type = get_type(expr->left);
609         if (!type || type->type != SYM_ARRAY)
610                 return 0;

612         return 1;
613 }

615 struct expression *get_array_base(struct expression *expr)
616 {
617         if (!is_array(expr))
618                 return NULL;
619         expr = strip_expr(expr);
620         if (expr->type == EXPR_PREOP && expr->op == '*')
621                 expr = strip_expr(expr->unop);
622         if (expr->type != EXPR_BINOP || expr->op != '+')
623                 return NULL;
624         return strip_parens(expr->left);
625 }

627 struct expression *get_array_offset(struct expression *expr)
628 {
629         if (!is_array(expr))
630                 return NULL;
631         expr = strip_expr(expr);
632         if (expr->type == EXPR_PREOP && expr->op == '*')
633                 expr = strip_expr(expr->unop);
634         if (expr->type != EXPR_BINOP || expr->op != '+')
635                 return NULL;
636         return strip_parens(expr->right);
637 }

639 const char *show_state(struct smatch_state *state)
640 {
641         if (!state)
642                 return NULL;
643         return state->name;
644 }

646 struct statement *get_expression_statement(struct expression *expr)
647 {
648         /* What are those things called? if (({....; ret;})) { ...*/

650         if (expr->type != EXPR_PREOP)
651                 return NULL;
652         if (expr->op != '(')
653                 return NULL;
654         if (!expr->unop)
```

```
655                 return NULL;
656         if (expr->unop->type != EXPR_STATEMENT)
657                 return NULL;
658         if (expr->unop->statement->type != STMT_COMPOUND)
659                 return NULL;
660         return expr->unop->statement;
661 }
663 struct expression *strip_parens(struct expression *expr)
664 {
665         if (!expr)
666                 return NULL;
668         if (expr->type == EXPR_PREOP) {
669                 if (!expr->unop)
670                         return expr;  /* parsing invalid code */
672                 if (expr->op == '(' && expr->unop->type == EXPR_STATEMENT &&
673                         expr->unop->statement->type == STMT_COMPOUND)
674                         return expr;
675                 if (expr->op == '(')
676                         return strip_parens(expr->unop);
677         }
678         return expr;
679 }
681 static struct expression *strip_expr_helper(struct expression *expr, bool set_pa
682 {
683         if (!expr)
684                 return NULL;
686         switch (expr->type) {
687         case EXPR_FORCE_CAST:
688         case EXPR_CAST:
689                 if (set_parent)
690                         expr_set_parent_expr(expr->cast_expression, expr);
692                 if (!expr->cast_expression)
693                         return expr;
694                 return strip_expr_helper(expr->cast_expression, set_parent);
695         case EXPR_PREOP: {
696                 struct expression *unop;
698                 if (!expr->unop)  /* parsing invalid code */
699                         return expr;
700                 if (set_parent)
701                         expr_set_parent_expr(expr->unop, expr);
704                 if (expr->op == '(' && expr->unop->type == EXPR_STATEMENT &&
705                         expr->unop->statement->type == STMT_COMPOUND)
706                         return expr;
708                 unop = strip_expr_helper(expr->unop, set_parent);
710                 if (expr->op == '*' && unop &&
711                     unop->type == EXPR_PREOP && unop->op == '&') {
712                         struct symbol *type = get_type(unop->unop);
714                         if (type && type->type == SYM_ARRAY)
715                                 return expr;
716                         return strip_expr_helper(unop->unop, set_parent);
717                 }
719                 if (expr->op == '(')
720                         return unop;
```



```
722                 return expr;
723         }
724         case EXPR_CONDITIONAL:
725                 if (known_condition_true(expr->conditional)) {
726                         if (expr->cond_true) {
727                                 if (set_parent)
728                                         expr_set_parent_expr(expr->cond_true, ex
729                                 return strip_expr_helper(expr->cond_true, set_pa
730                         }
731                         if (set_parent)
732                                 expr_set_parent_expr(expr->conditional, expr);
733                         return strip_expr_helper(expr->conditional, set_parent);
734                 }
735                 if (known_condition_false(expr->conditional)) {
736                         if (set_parent)
737                                 expr_set_parent_expr(expr->cond_false, expr);
738                         return strip_expr_helper(expr->cond_false, set_parent);
739                 }
740                 return expr;
741         case EXPR_CALL:
742                 if (sym_name_is("__builtin_expect", expr->fn) ||
743                     sym_name_is("__builtin_bswap16", expr->fn) ||
744                     sym_name_is("__builtin_bswap32", expr->fn) ||
745                     sym_name_is("__builtin_bswap64", expr->fn)) {
746                         expr = get_argument_from_call_expr(expr->args, 0);
747                         return strip_expr_helper(expr, set_parent);
748                 }
749                 return expr;
750         }
751         return expr;
752 }
754 struct expression *strip_expr(struct expression *expr)
755 {
756         return strip_expr_helper(expr, false);
757 }
759 struct expression *strip_expr_set_parent(struct expression *expr)
760 {
761         return strip_expr_helper(expr, true);
762 }
764 static void delete_state_tracker(struct tracker *t)
765 {
766         delete_state(t->owner, t->name, t->sym);
767         __free_tracker(t);
768 }
770 void scoped_state(int my_id, const char *name, struct symbol *sym)
771 {
772         struct tracker *t;
774         t = alloc_tracker(my_id, name, sym);
775         add_scope_hook((scope_hook *)&delete_state_tracker, t);
776 }
778 int is_error_return(struct expression *expr)
779 {
780         struct symbol *cur_func = cur_func_sym;
781         struct range_list *rl;
782         sval_t sval;
784         if (!expr)
785                 return 0;
786         if (cur_func->type != SYM_NODE)
```

```
787                 return 0;
788         cur_func = get_base_type(cur_func);
789         if (cur_func->type != SYM_FN)
790                 return 0;
791         cur_func = get_base_type(cur_func);
792         if (cur_func == &void_ctype)
793                 return 0;
794         if (option_project == PROJ_KERNEL &&
795             get_implied_rl(expr, &rl) &&
796             rl_type(rl) == &int_ctype &&
797             sval_is_negative(rl_min(rl)) &&
798             rl_max(rl).value == -1)
799                 return 1;
800         if (!get_implied_value(expr, &sval))
801                 return 0;
802         if (sval.value < 0)
803                 return 1;
804         if (cur_func->type == SYM_PTR && sval.value == 0)
805                 return 1;
806         return 0;
807 }

809 int getting_address(void)
810 {
811         struct expression *tmp;
812         int i = 0;
813         int dot_ops = 0;

815         FOR_EACH_PTR_REVERSE(big_expression_stack, tmp) {
816                 if (!i++)
817                         continue;
818                 if (tmp->type == EXPR_PREOP && tmp->op == '(')
819                         continue;
820                 if (tmp->op == '.' && !dot_ops++)
821                         continue;
822                 if (tmp->op == '&')
823                         return 1;
824                 return 0;
825         } END_FOR_EACH_PTR_REVERSE(tmp);
826         return 0;
827 }

829 int get_struct_and_member(struct expression *expr, const char **type, const char
830 {
831         struct symbol *sym;

833         expr = strip_expr(expr);
834         if (expr->type != EXPR_DEREF)
835                 return 0;
836         if (!expr->member)
837                 return 0;

839         sym = get_type(expr->deref);
840         if (!sym)
841                 return 0;
842         if (sym->type == SYM_UNION)
843                 return 0;
844         if (!sym->ident)
845                 return 0;

847         *type = sym->ident->name;
848         *member = expr->member->name;
849         return 1;
850 }

852 char *get_member_name(struct expression *expr)
```



```
853 {
854         char buf[256];
855         struct symbol *sym;

857         expr = strip_expr(expr);
858         if (!expr || expr->type != EXPR_DEREF)
859                 return NULL;
860         if (!expr->member)
861                 return NULL;

863         sym = get_type(expr->deref);
864         if (!sym)
865                 return NULL;
866         if (sym->type == SYM_UNION) {
867                 snprintf(buf, sizeof(buf), "(union %s)->%s",
868                          sym->ident ? sym->ident->name : "anonymous",
869                          expr->member->name);
870                 return alloc_string(buf);
871         }
872         if (!sym->ident)
873                 return NULL;
874         snprintf(buf, sizeof(buf), "(struct %s)->%s", sym->ident->name, expr->me
875         return alloc_string(buf);
876 }

878 int cmp_pos(struct position pos1, struct position pos2)
879 {
880         /* the stream position is ... */
881         if (pos1.stream > pos2.stream)
882                 return -1;
883         if (pos1.stream < pos2.stream)
884                 return 1;

886         if (pos1.line < pos2.line)
887                 return -1;
888         if (pos1.line > pos2.line)
889                 return 1;

891         if (pos1.pos < pos2.pos)
892                 return -1;
893         if (pos1.pos > pos2.pos)
894                 return 1;

896         return 0;
897 }

899 int positions_eq(struct position pos1, struct position pos2)
900 {
901         if (pos1.line != pos2.line)
902                 return 0;
903         if (pos1.pos != pos2.pos)
904                 return 0;
905         if (pos1.stream != pos2.stream)
906                 return 0;
907         return 1;
908 }

910 struct statement *get_current_statement(void)
911 {
912         struct statement *prev, *tmp;

914         prev = last_ptr_list((struct ptr_list *)big_statement_stack);

916         if (!prev || !get_macro_name(prev->pos))
917                 return prev;
```

```
919         FOR_EACH_PTR_REVERSE(big_statement_stack, tmp) {
920                 if (positions_eq(tmp->pos, prev->pos))
921                         continue;
922                 if (prev->pos.line > tmp->pos.line)
923                         return prev;
924                 return tmp;
925         } END_FOR_EACH_PTR_REVERSE(tmp);
926         return prev;
927 }

929 struct statement *get_prev_statement(void)
930 {
931         struct statement *tmp;
932         int i;

934         i = 0;
935         FOR_EACH_PTR_REVERSE(big_statement_stack, tmp) {
936                 if (i++ == 1)
937                         return tmp;
938         } END_FOR_EACH_PTR_REVERSE(tmp);
939         return NULL;
940 }

942 struct expression *get_last_expr_from_expression_stmt(struct expression *expr)
943 {
944         struct statement *stmt;
945         struct statement *last_stmt;

947         while (expr->type == EXPR_PREOP && expr->op == '(')
948                 expr = expr->unop;
949         if (expr->type != EXPR_STATEMENT)
950                 return NULL;
951         stmt = expr->statement;
952         if (!stmt)
953                 return NULL;
954         if (stmt->type == STMT_COMPOUND) {
955                 last_stmt = last_ptr_list((struct ptr_list *)stmt->stmts);
956                 if (!last_stmt)
957                         return NULL;
958                 if (last_stmt->type == STMT_LABEL)
959                         last_stmt = last_stmt->label_statement;
960                 if (last_stmt->type != STMT_EXPRESSION)
961                         return NULL;
962                 return last_stmt->expression;
963         }
964         if (stmt->type == STMT_EXPRESSION)
965                 return stmt->expression;
966         return NULL;
967 }

969 int get_param_num_from_sym(struct symbol *sym)
970 {
971         struct symbol *tmp;
972         int i;

974         if (!cur_func_sym)
975                 return -1;

977         i = 0;
978         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, tmp) {
979                 if (tmp == sym)
980                         return i;
981                 i++;
982         } END_FOR_EACH_PTR(tmp);
983         return -1;
984 }
```



```
986 int get_param_num(struct expression *expr)
987 {
988         struct symbol *sym;
989         char *name;

991         if (!cur_func_sym)
992                 return -1;
993         name = expr_to_var_sym(expr, &sym);
994         free_string(name);
995         if (!sym)
996                 return -1;
997         return get_param_num_from_sym(sym);
998 }

1000 int ms_since(struct timeval *start)
1001 {
1002         struct timeval end;
1003         double diff;

1005         gettimeofday(&end, NULL);
1006         diff = (end.tv_sec - start->tv_sec) * 1000.0;
1007         diff += (end.tv_usec - start->tv_usec) / 1000.0;
1008         return (int)diff;
1009 }

1011 int parent_is_gone_var_sym(const char *name, struct symbol *sym)
1012 {
1013         if (!name || !sym)
1014                 return 0;

1016         if (parent_is_null_var_sym(name, sym) ||
1017             parent_is_free_var_sym(name, sym))
1018                 return 1;
1019         return 0;
1020 }

1022 int parent_is_gone(struct expression *expr)
1023 {
1024         struct symbol *sym;
1025         char *var;
1026         int ret = 0;

1028         expr = strip_expr(expr);
1029         var = expr_to_var_sym(expr, &sym);
1030         if (!var || !sym)
1031                 goto free;
1032         ret = parent_is_gone_var_sym(var, sym);
1033 free:
1034         free_string(var);
1035         return ret;
1036 }

1038 int invert_op(int op)
1039 {
1040         switch (op) {
1041         case '*':
1042                 return '/';
1043         case '/':
1044                 return '*';
1045         case '+':
1046                 return '-';
1047         case '-':
1048                 return '+';
1049         case SPECIAL_LEFTSHIFT:
1050                 return SPECIAL_RIGHTSHIFT;
```

```
1051         case SPECIAL_RIGHTSHIFT:
1052                 return SPECIAL_LEFTSHIFT;
1053         }
1054         return 0;
1055 }

1057 int expr_equiv(struct expression *one, struct expression *two)
1058 {
1059         struct symbol *one_sym = NULL;
1060         struct symbol *two_sym = NULL;
1061         char *one_name = NULL;
1062         char *two_name = NULL;
1063         int ret = 0;

1065         if (!one || !two)
1066                 return 0;
1067         if (one->type != two->type)
1068                 return 0;
1069         if (is_fake_call(one) || is_fake_call(two))
1070                 return 0;

1072         one_name = expr_to_str_sym(one, &one_sym);
1073         if (!one_name)
1074                 goto free;
1075         two_name = expr_to_str_sym(two, &two_sym);
1076         if (!two_name)
1077                 goto free;
1078         if (one_sym != two_sym)
1079                 goto free;
1080         /*
1081          * This is a terrible hack because expr_to_str() sometimes gives up in
1082          * the middle and just returns what it has.  If you see a () you know
1083          * the string is bogus.
1084          */
1085         if (strstr(one_name, "()"))
1086                 goto free;
1087         if (strcmp(one_name, two_name) == 0)
1088                 ret = 1;
1089 free:
1090         free_string(one_name);
1091         free_string(two_name);
1092         return ret;
1093 }

1095 void push_int(struct int_stack **stack, int num)
1096 {
1097         int *munged;

1099         /*
1100          * Just put the int on directly instead of a pointer to the int.
1101          * Shift it to the left because Sparse uses the last two bits.
1102          * This is sort of a dirty hack, yes.
1103          */

1105         munged = INT_PTR(num << 2);

1107         add_ptr_list(stack, munged);
1108 }

1110 int pop_int(struct int_stack **stack)
1111 {
1112         int *num;

1114         num = last_ptr_list((struct ptr_list *)*stack);
1115         delete_ptr_list_last((struct ptr_list **)stack);
```



```
1117         return PTR_INT(num) >> 2;
1118 }
```

```
**********************************************************
    8854 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_hooks.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2006 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

enum data_type {
        EXPR_PTR,
        STMT_PTR,
        SYMBOL_PTR,
        SYM_LIST_PTR,
};

struct hook_container {
        int hook_type;
        enum data_type data_type;
        void *fn;
};
ALLOCATOR(hook_container, "hook functions");
DECLARE_PTR_LIST(hook_func_list, struct hook_container);
static struct hook_func_list *merge_funcs;
static struct hook_func_list *unmatched_state_funcs;
static struct hook_func_list *hook_array[NUM_HOOKS] = {};
void (**pre_merge_hooks)(struct sm_state *sm);

struct scope_container {
        void *fn;
        void *data;
};
ALLOCATOR(scope_container, "scope hook functions");
DECLARE_PTR_LIST(scope_hook_list, struct scope_container);
DECLARE_PTR_LIST(scope_hook_stack, struct scope_hook_list);
static struct scope_hook_stack *scope_hooks;

void add_hook(void *func, enum hook_type type)
{
        struct hook_container *container = __alloc_hook_container(0);

        container->hook_type = type;
        container->fn = func;
        switch (type) {
        case EXPR_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case STMT_HOOK:
                container->data_type = STMT_PTR;
                break;
```



```
        case STMT_HOOK_AFTER:
                container->data_type = STMT_PTR;
                break;
        case SYM_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case STRING_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case DECLARATION_HOOK:
                container->data_type = SYMBOL_PTR;
                break;
        case ASSIGNMENT_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case ASSIGNMENT_HOOK_AFTER:
                container->data_type = EXPR_PTR;
                break;
        case RAW_ASSIGNMENT_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case GLOBAL_ASSIGNMENT_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case CALL_ASSIGNMENT_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case MACRO_ASSIGNMENT_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case BINOP_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case OP_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case LOGIC_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case PRELOOP_HOOK:
                container->data_type = STMT_PTR;
                break;
        case CONDITION_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case SELECT_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case WHOLE_CONDITION_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case FUNCTION_CALL_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case CALL_HOOK_AFTER_INLINE:
                container->data_type = EXPR_PTR;
                break;
        case FUNCTION_CALL_HOOK_AFTER_DB:
                container->data_type = EXPR_PTR;
                break;
        case DEREF_HOOK:
                container->data_type = EXPR_PTR;
                break;
        case CASE_HOOK:
                /* nothing needed */
                break;
```

```
127         case ASM_HOOK:
128                 container->data_type = STMT_PTR;
129                 break;
130         case CAST_HOOK:
131                 container->data_type = EXPR_PTR;
132                 break;
133         case SIZEOF_HOOK:
134                 container->data_type = EXPR_PTR;
135                 break;
136         case BASE_HOOK:
137                 container->data_type = SYMBOL_PTR;
138                 break;
139         case FUNC_DEF_HOOK:
140                 container->data_type = SYMBOL_PTR;
141                 break;
142         case AFTER_DEF_HOOK:
143                 container->data_type = SYMBOL_PTR;
144                 break;
145         case END_FUNC_HOOK:
146                 container->data_type = SYMBOL_PTR;
147                 break;
148         case AFTER_FUNC_HOOK:
149                 container->data_type = SYMBOL_PTR;
150                 break;
151         case RETURN_HOOK:
152                 container->data_type = EXPR_PTR;
153                 break;
154         case INLINE_FN_START:
155                 container->data_type = EXPR_PTR;
156                 break;
157         case INLINE_FN_END:
158                 container->data_type = EXPR_PTR;
159                 break;
160         case END_FILE_HOOK:
161                 container->data_type = SYM_LIST_PTR;
162                 break;
163         }
164         add_ptr_list(&hook_array[type], container);
165 }

167 void add_merge_hook(int client_id, merge_func_t *func)
168 {
169         struct hook_container *container = __alloc_hook_container(0);
170         container->data_type = client_id;
171         container->fn = func;
172         add_ptr_list(&merge_funcs, container);
173 }

175 void add_unmatched_state_hook(int client_id, unmatched_func_t *func)
176 {
177         struct hook_container *container = __alloc_hook_container(0);
178         container->data_type = client_id;
179         container->fn = func;
180         add_ptr_list(&unmatched_state_funcs, container);
181 }

183 void add_pre_merge_hook(int client_id, void (*hook)(struct sm_state *sm))
184 {
185         pre_merge_hooks[client_id] = hook;
186 }

188 static void pass_to_client(void *fn)
189 {
190         typedef void (expr_func)();
191         ((expr_func *) fn)();
192 }
```



```
194 static void pass_expr_to_client(void *fn, void *data)
195 {
196         typedef void (expr_func)(struct expression *expr);
197         ((expr_func *) fn)((struct expression *) data);
198 }

200 static void pass_stmt_to_client(void *fn, void *data)
201 {
202         typedef void (stmt_func)(struct statement *stmt);
203         ((stmt_func *) fn)((struct statement *) data);
204 }

206 static void pass_sym_to_client(void *fn, void *data)
207 {
208         typedef void (sym_func)(struct symbol *sym);
209         ((sym_func *) fn)((struct symbol *) data);
210 }

212 static void pass_sym_list_to_client(void *fn, void *data)
213 {
214         typedef void (sym_func)(struct symbol_list *sym_list);
215         ((sym_func *) fn)((struct symbol_list *) data);
216 }

218 void __pass_to_client(void *data, enum hook_type type)
219 {
220         struct hook_container *container;


223         FOR_EACH_PTR(hook_array[type], container) {
224                 switch (container->data_type) {
225                 case EXPR_PTR:
226                         pass_expr_to_client(container->fn, data);
227                         break;
228                 case STMT_PTR:
229                         pass_stmt_to_client(container->fn, data);
230                         break;
231                 case SYMBOL_PTR:
232                         pass_sym_to_client(container->fn, data);
233                         break;
234                 case SYM_LIST_PTR:
235                         pass_sym_list_to_client(container->fn, data);
236                         break;
237                 }
238         } END_FOR_EACH_PTR(container);
239 }

241 void __pass_to_client_no_data(enum hook_type type)
242 {
243         struct hook_container *container;

245         FOR_EACH_PTR(hook_array[type], container) {
246                 pass_to_client(container->fn);
247         } END_FOR_EACH_PTR(container);
248 }

250 void __pass_case_to_client(struct expression *switch_expr,
251                            struct range_list *rl)
252 {
253         typedef void (case_func)(struct expression *switch_expr,
254                                  struct range_list *rl);
255         struct hook_container *container;

257         FOR_EACH_PTR(hook_array[CASE_HOOK], container) {
258                 ((case_func *) container->fn)(switch_expr, rl);
```

```
259         } END_FOR_EACH_PTR(container);
260 }

262 int __has_merge_function(int client_id)
263 {
264         struct hook_container *tmp;

266         FOR_EACH_PTR(merge_funcs, tmp) {
267                 if (tmp->data_type == client_id)
268                         return 1;
269         } END_FOR_EACH_PTR(tmp);
270         return 0;
271 }

273 struct smatch_state *__client_merge_function(int owner,
274                                              struct smatch_state *s1,
275                                              struct smatch_state *s2)
276 {
277         struct smatch_state *tmp_state;
278         struct hook_container *tmp;

280         /* Pass NULL states first and the rest alphabetically by name */
281         if (!s2 || (s1 && strcmp(s2->name, s1->name) < 0)) {
282                 tmp_state = s1;
283                 s1 = s2;
284                 s2 = tmp_state;
285         }

287         FOR_EACH_PTR(merge_funcs, tmp) {
288                 if (tmp->data_type == owner)
289                         return ((merge_func_t *) tmp->fn)(s1, s2);
290         } END_FOR_EACH_PTR(tmp);
291         return &undefined;
292 }

294 struct smatch_state *__client_unmatched_state_function(struct sm_state *sm)
295 {
296         struct hook_container *tmp;

298         FOR_EACH_PTR(unmatched_state_funcs, tmp) {
299                 if (tmp->data_type == sm->owner)
300                         return ((unmatched_func_t *) tmp->fn)(sm);
301         } END_FOR_EACH_PTR(tmp);
302         return &undefined;
303 }

305 void call_pre_merge_hook(struct sm_state *sm)
306 {
307         if (sm->owner >= num_checks)
308                 return;

310         if (pre_merge_hooks[sm->owner])
311                 pre_merge_hooks[sm->owner](sm);
312 }

314 static struct scope_hook_list *pop_scope_hook_list(struct scope_hook_stack **sta
315 {
316         struct scope_hook_list *hook_list;

318         hook_list = last_ptr_list((struct ptr_list *)*stack);
319         delete_ptr_list_last((struct ptr_list **)stack);
320         return hook_list;
321 }

323 static void push_scope_hook_list(struct scope_hook_stack **stack, struct scope_h
324 {
```



```
325         add_ptr_list(stack, l);
326 }

328 void add_scope_hook(scope_hook *fn, void *data)
329 {
330         struct scope_hook_list *hook_list;
331         struct scope_container *new;

333         if (!scope_hooks)
334                 return;
335         hook_list = pop_scope_hook_list(&scope_hooks);
336         new = __alloc_scope_container(0);
337         new->fn = fn;
338         new->data = data;
339         add_ptr_list(&hook_list, new);
340         push_scope_hook_list(&scope_hooks, hook_list);
341 }

343 void __push_scope_hooks(void)
344 {
345         push_scope_hook_list(&scope_hooks, NULL);
346 }

348 void __call_scope_hooks(void)
349 {
350         struct scope_hook_list *hook_list;
351         struct scope_container *tmp;

353         if (!scope_hooks)
354                 return;

356         hook_list = pop_scope_hook_list(&scope_hooks);
357         FOR_EACH_PTR(hook_list, tmp) {
358                 ((scope_hook *) tmp->fn)(tmp->data);
359                 __free_scope_container(tmp);
360         } END_FOR_EACH_PTR(tmp);
361 }

363 void allocate_hook_memory(void)
364 {
365         pre_merge_hooks = malloc(num_checks * sizeof(*pre_merge_hooks));
366         memset(pre_merge_hooks, 0, num_checks * sizeof(*pre_merge_hooks));
367 }
```

```
**********************************************************
    1670 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_ignore.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

STATE(ignore);
static struct stree *ignored;

void add_ignore(int owner, const char *name, struct symbol *sym)
{
        set_state_stree(&ignored, owner, name, sym, &ignore);
}

int is_ignored(int owner, const char *name, struct symbol *sym)
{
        return !!get_state_stree(ignored, owner, name, sym);
}

void add_ignore_expr(int owner, struct expression *expr)
{
        struct symbol *sym;
        char *name;

        name = expr_to_str_sym(expr, &sym);
        if (!name || !sym)
                return;
        add_ignore(owner, name, sym);
        free_string(name);
}

int is_ignored_expr(int owner, struct expression *expr)
{
        struct symbol *sym;
        char *name;
        int ret;

        name = expr_to_str_sym(expr, &sym);
        if (!name && !sym)
                return 0;
        ret = is_ignored(owner, name, sym);
        free_string(name);
        return ret;
}

static void clear_ignores(void)
```



```
{
        if (__inline_fn)
                return;
        free_stree(&ignored);
}

void register_smatch_ignore(int id)
{
        add_hook(&clear_ignores, AFTER_FUNC_HOOK);
}
```

```
**********************************************************
    2187 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_imaginary_absolute.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2016 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Say you have a condition like:
 *
 *     if (foo < 0)
 *         return foo;
 *
 * But we actually know that foo is zero.  Then in smatch_extra.c we set "foo"
 * to the empty state and then for the return statement we say that "foo" is
 * s32min-s32max because we can't return the empty state.
 *
 * This file is supposed to provide an alternative to say that actually "foo" is
 * less than zero.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static struct smatch_state *empty_state(struct sm_state *sm)
{
        return alloc_estate_empty();
}

struct smatch_state *merge_is_empty(struct smatch_state *s1, struct smatch_state
{
        return alloc_estate_empty();
}

static void reset(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, alloc_estate_empty());
}

void __save_imaginary_state(struct expression *expr, struct range_list *true_rl,
{
        set_true_false_states_expr(my_id, expr, alloc_estate_rl(true_rl), alloc_
}

int get_imaginary_absolute(struct expression *expr, struct range_list **rl)
{
```



```
        struct smatch_state *state;

        *rl = NULL;

        state = get_state_expr(my_id, expr);
        if (!state || !estate_rl(state))
                return 0;

        *rl = estate_rl(state);
        return 1;
}

void register_imaginary_absolute(int id)
{
        my_id = id;

        add_unmatched_state_hook(my_id, &empty_state);
        add_merge_hook(my_id, &merge_is_empty);
        add_modification_hook(my_id, &reset);
}
```

```
**********************************************************
   30846 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_implied.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2008 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Imagine we have this code:
 * foo = 1;
 * if (bar)
 *         foo = 99;
 * else
 *         frob();
 *                   //  <-- point #1
 * if (foo == 99)    //  <-- point #2
 *         bar->baz; //  <-- point #3
 *
 *
 * At point #3 bar is non null and can be dereferenced.
 *
 * It's smatch_implied.c which sets bar to non null at point #2.
 *
 * At point #1 merge_slist() stores the list of states from both
 * the true and false paths.  On the true path foo == 99 and on
 * the false path foo == 1.  merge_slist() sets their pool
 * list to show the other states which were there when foo == 99.
 *
 * When it comes to the if (foo == 99) the smatch implied hook
 * looks for all the pools where foo was not 99.  It makes a list
 * of those.
 *
 * Then for bar (and all the other states) it says, ok bar is a
 * merged state that came from these previous states.  We'll
 * chop out all the states where it came from a pool where
 * foo != 99 and merge it all back together.
 *
 * That is the implied state of bar.
 *
 * merge_slist() sets up ->pool.  An sm_state only has one ->pool and
 *    that is the pool where it was first set.  The my pool gets set when
 *    code paths merge.  States that have been set since the last merge do
 *    not have a ->pool.
 * merge_sm_state() sets ->left and ->right.  (These are the states which were
 *    merged to form the current state.)
 * a pool:  a pool is an slist that has been merged with another slist.
 */

#include <sys/time.h>
#include <time.h>
```



```
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

char *implied_debug_msg;
#define DIMPLIED(msg...) do { if (option_debug_implied || option_debug) printf(m

int option_debug_implied = 0;

/*
 * tmp_range_list():
 * It messes things up to free range list allocations.  This helper fuction
 * lets us reuse memory instead of doing new allocations.
 */
static struct range_list *tmp_range_list(struct symbol *type, long long num)
{
        static struct range_list *my_list = NULL;
        static struct data_range *my_range;

        __free_ptr_list((struct ptr_list **)&my_list);
        my_range = alloc_range(ll_to_sval(num), ll_to_sval(num));
        add_ptr_list(&my_list, my_range);
        return my_list;
}

static void print_debug_tf(struct sm_state *sm, int istrue, int isfalse)
{
        if (!option_debug_implied && !option_debug)
                return;

        if (istrue && isfalse) {
                printf("%s: %d: does not exist.\n", show_sm(sm), sm->line);
        } else if (istrue) {
                printf("'%s = %s' from %d is true. %s[stree %d]\n", sm->name, sh
                        sm->line, sm->merged ? "[merged]" : "[leaf]",
                        get_stree_id(sm->pool));
        } else if (isfalse) {
                printf("'%s = %s' from %d is false. %s[stree %d]\n", sm->name, s
                        sm->line,
                        sm->merged ? "[merged]" : "[leaf]",
                        get_stree_id(sm->pool));
        } else {
                printf("'%s = %s' from %d could be true or false. %s[stree %d]\n
                        show_state(sm->state), sm->line,
                        sm->merged ? "[merged]" : "[leaf]",
                        get_stree_id(sm->pool));
        }
}

static int create_fake_history(struct sm_state *sm, int comparison, struct range
{
        struct range_list *orig_rl;
        struct range_list *true_rl, *false_rl;
        struct stree *true_stree, *false_stree;
        struct sm_state *true_sm, *false_sm;
        sval_t sval;

        if (is_merged(sm) || sm->left || sm->right)
                return 0;
        if (!rl_to_sval(rl, &sval))
                return 0;
        if (!estate_rl(sm->state))
                return 0;

        orig_rl = cast_rl(rl_type(rl), estate_rl(sm->state));
        split_comparison_rl(orig_rl, comparison, rl, &true_rl, &false_rl, NULL,
```

```
128         true_rl = rl_truncate_cast(estate_type(sm->state), true_rl);
129         false_rl = rl_truncate_cast(estate_type(sm->state), false_rl);
130         if (is_whole_rl(true_rl) || is_whole_rl(false_rl) ||
131             !true_rl || !false_rl ||
132             rl_equiv(orig_rl, true_rl) || rl_equiv(orig_rl, false_rl) ||
133             rl_equiv(estate_rl(sm->state), true_rl) || rl_equiv(estate_rl(sm->st
134                 return 0;

136         if (rl_intersection(true_rl, false_rl)) {
137                 sm_perror("parsing (%s (%s) %s %s)",
138                           sm->name, sm->state->name, show_special(comparison), sho
139                 sm_msg("true_rl = %s false_rl = %s intersection = %s",
140                        show_rl(true_rl), show_rl(false_rl), show_rl(rl_intersect
141                 return 0;
142         }

144         if (option_debug)
145                 sm_info("fake_history: %s vs %s.  %s %s %s. --> T: %s F: %s",
146                        sm->name, show_rl(rl), sm->state->name, show_special(comp
147                        show_rl(true_rl), show_rl(false_rl));

149         true_sm = clone_sm(sm);
150         false_sm = clone_sm(sm);

152         true_sm->state = alloc_estate_rl(cast_rl(estate_type(sm->state), true_rl
153         free_slist(&true_sm->possible);
154         add_possible_sm(true_sm, true_sm);
155         false_sm->state = alloc_estate_rl(cast_rl(estate_type(sm->state), false_
156         free_slist(&false_sm->possible);
157         add_possible_sm(false_sm, false_sm);

159         true_stree = clone_stree(sm->pool);
160         false_stree = clone_stree(sm->pool);

162         overwrite_sm_state_stree(&true_stree, true_sm);
163         overwrite_sm_state_stree(&false_stree, false_sm);

165         true_sm->pool = true_stree;
166         false_sm->pool = false_stree;

168         sm->merged = 1;
169         sm->left = true_sm;
170         sm->right = false_sm;

172         return 1;
173 }

175 /*
176  * add_pool() adds a slist to *pools. If the slist has already been
177  * added earlier then it doesn't get added a second time.
178  */
179 void add_pool(struct state_list **pools, struct sm_state *new)
180 {
181         struct sm_state *tmp;

183         FOR_EACH_PTR(*pools, tmp) {
184                 if (tmp->pool < new->pool)
185                         continue;
186                 else if (tmp->pool == new->pool) {
187                         return;
188                 } else {
189                         INSERT_CURRENT(new, tmp);
190                         return;
191                 }
192         } END_FOR_EACH_PTR(tmp);
```



```
193         add_ptr_list(pools, new);
194 }

196 static int pool_in_pools(struct stree *pool,
197                          const struct state_list *slist)
198 {
199         struct sm_state *tmp;

201         FOR_EACH_PTR(slist, tmp) {
202                 if (!tmp->pool)
203                         continue;
204                 if (tmp->pool == pool)
205                         return 1;
206         } END_FOR_EACH_PTR(tmp);
207         return 0;
208 }

210 static int remove_pool(struct state_list **pools, struct stree *remove)
211 {
212         struct sm_state *tmp;
213         int ret = 0;

215         FOR_EACH_PTR(*pools, tmp) {
216                 if (tmp->pool == remove) {
217                         DELETE_CURRENT_PTR(tmp);
218                         ret = 1;
219                 }
220         } END_FOR_EACH_PTR(tmp);

222         return ret;
223 }

225 /*
226  * If 'foo' == 99 add it that pool to the true pools.  If it's false, add it to
227  * the false pools.  If we're not sure, then we don't add it to either.
228  */
229 static void do_compare(struct sm_state *sm, int comparison, struct range_list *r
230                        struct state_list **true_stack,
231                        struct state_list **maybe_stack,
232                        struct state_list **false_stack,
233                        int *mixed, struct sm_state *gate_sm)
234 {
235         int istrue;
236         int isfalse;
237         struct range_list *var_rl;

239         if (!sm->pool)
240                 return;

242         var_rl = cast_rl(rl_type(rl), estate_rl(sm->state));

244         istrue = !possibly_false_rl(var_rl, comparison, rl);
245         isfalse = !possibly_true_rl(var_rl, comparison, rl);

247         print_debug_tf(sm, istrue, isfalse);

249         /* give up if we have borrowed implications (smatch_equiv.c) */
250         if (sm->sym != gate_sm->sym ||
251             strcmp(sm->name, gate_sm->name) != 0) {
252                 if (mixed)
253                         *mixed = 1;
254         }

256         if (mixed && !*mixed && !is_merged(sm) && !istrue && !isfalse) {
257                 if (!create_fake_history(sm, comparison, rl))
258                         *mixed = 1;
```

```
259         }
260
261         if (istrue)
262                 add_pool(true_stack, sm);
263         else if (isfalse)
264                 add_pool(false_stack, sm);
265         else
266                 add_pool(maybe_stack, sm);
267
268 }
269
270 static int is_checked(struct state_list *checked, struct sm_state *sm)
271 {
272         struct sm_state *tmp;
273
274         FOR_EACH_PTR(checked, tmp) {
275                 if (tmp == sm)
276                         return 1;
277         } END_FOR_EACH_PTR(tmp);
278         return 0;
279 }
280
281 /*
282  * separate_pools():
283  * Example code:  if (foo == 99) {
284  *
285  * Say 'foo' is a merged state that has many possible values.  It is the combina
286  * of merges.  separate_pools() iterates through the pools recursively and calls
287  * do_compare() for each time 'foo' was set.
288  */
289 static void __separate_pools(struct sm_state *sm, int comparison, struct range_l
290                         struct state_list **true_stack,
291                         struct state_list **maybe_stack,
292                         struct state_list **false_stack,
293                         struct state_list **checked, int *mixed, struct sm_state
294 {
295         int free_checked = 0;
296         struct state_list *checked_states = NULL;
297
298         if (!sm)
299                 return;
300
301         /*
302          * If it looks like this is going to take too long as-is, then don't
303          * create even more fake history.
304          */
305         if (mixed && sm->nr_children > 100)
306                 *mixed = 1;
307
308         /*
309            Sometimes the implications are just too big to deal with
310            so we bail.  Theoretically, bailing out here can cause more false
311            positives but won't hide actual bugs.
312         */
313         if (sm->nr_children > 4000) {
314                 if (option_debug || option_debug_implied) {
315                         static char buf[1028];
316                         snprintf(buf, sizeof(buf), "debug: %s: nr_children over
317                                 __func__, sm->nr_children, sm->name, show_state
318                         implied_debug_msg = buf;
319                 }
320                 return;
321         }
322
323         if (checked == NULL) {
324                 checked = &checked_states;
```



```
325                 free_checked = 1;
326         }
327         if (is_checked(*checked, sm))
328                 return;
329         add_ptr_list(checked, sm);
330
331         do_compare(sm, comparison, rl, true_stack, maybe_stack, false_stack, mix
332
333         __separate_pools(sm->left, comparison, rl, true_stack, maybe_stack, fals
334         __separate_pools(sm->right, comparison, rl, true_stack, maybe_stack, fal
335         if (free_checked)
336                 free_slist(checked);
337 }
338
339 static void separate_pools(struct sm_state *sm, int comparison, struct range_lis
340                         struct state_list **true_stack,
341                         struct state_list **false_stack,
342                         struct state_list **checked, int *mixed)
343 {
344         struct state_list *maybe_stack = NULL;
345         struct sm_state *tmp;
346
347         __separate_pools(sm, comparison, rl, true_stack, &maybe_stack, false_sta
348
349         if (option_debug) {
350                 struct sm_state *sm;
351
352                 FOR_EACH_PTR(*true_stack, sm) {
353                         sm_msg("TRUE %s [stree %d]", show_sm(sm), get_stree_id(s
354                 } END_FOR_EACH_PTR(sm);
355
356                 FOR_EACH_PTR(maybe_stack, sm) {
357                         sm_msg("MAYBE %s [stree %d]", show_sm(sm), get_stree_id(
358                 } END_FOR_EACH_PTR(sm);
359
360                 FOR_EACH_PTR(*false_stack, sm) {
361                         sm_msg("FALSE %s [stree %d]", show_sm(sm), get_stree_id(
362                 } END_FOR_EACH_PTR(sm);
363         }
364         /* if it's a maybe then remove it */
365         FOR_EACH_PTR(maybe_stack, tmp) {
366                 remove_pool(false_stack, tmp->pool);
367                 remove_pool(true_stack, tmp->pool);
368         } END_FOR_EACH_PTR(tmp);
369
370         /* if it's both true and false remove it from both */
371         FOR_EACH_PTR(*true_stack, tmp) {
372                 if (remove_pool(false_stack, tmp->pool))
373                         DELETE_CURRENT_PTR(tmp);
374         } END_FOR_EACH_PTR(tmp);
375 }
376
377 static int sm_in_keep_leafs(struct sm_state *sm, const struct state_list *keep_g
378 {
379         struct sm_state *tmp, *old;
380
381         FOR_EACH_PTR(keep_gates, tmp) {
382                 if (is_merged(tmp))
383                         continue;
384                 old = get_sm_state_stree(tmp->pool, sm->owner, sm->name, sm->sym
385                 if (!old)
386                         continue;
387                 if (old == sm)
388                         return 1;
389         } END_FOR_EACH_PTR(tmp);
390         return 0;
```

```
391 }
393 static int taking_too_long(void)
394 {
395         static void *printed;
397         if (out_of_memory())
398                 return 1;
400         if (time_parsing_function() < 60)
401                 return 0;
403         if (!__inline_fn && printed != cur_func_sym) {
404                 if (!is_skipped_function())
405                         sm_perror("turning off implications after 60 seconds");
406                 printed = cur_func_sym;
407         }
408         return 1;
409 }
411 /*
412  * NOTE: If a state is in both the keep stack and the remove stack then that is
413  * a bug.  Only add states which are definitely true or definitely false.  If
414  * you have a leaf state that could be both true and false, then create a fake
415  * split history where one side is true and one side is false.  Otherwise, if
416  * you can't do that, then don't add it to either list.
417  */
418 struct sm_state *filter_pools(struct sm_state *sm,
419                               const struct state_list *remove_stack,
420                               const struct state_list *keep_stack,
421                               int *modified, int *recurse_cnt,
422                               struct timeval *start)
423 {
424         struct sm_state *ret = NULL;
425         struct sm_state *left;
426         struct sm_state *right;
427         int removed = 0;
428         struct timeval now;
430         if (!sm)
431                 return NULL;
432         if (sm->skip_implications)
433                 return sm;
434         if (taking_too_long())
435                 return sm;
437         gettimeofday(&now, NULL);
438         if ((*recurse_cnt)++ > 1000 || now.tv_sec - start->tv_sec > 5) {
439                 if (local_debug || option_debug_implied) {
440                         static char buf[1028];
441                         snprintf(buf, sizeof(buf), "debug: %s: nr_children over
442                                  __func__, sm->nr_children, sm->name, show_state
443                         implied_debug_msg = buf;
444                 }
445                 sm->skip_implications = 1;
446                 return sm;
447         }
449         if (pool_in_pools(sm->pool, remove_stack)) {
450                 DIMPLIED("removed [stree %d] %s from %d\n", get_stree_id(sm->poo
451                 *modified = 1;
452                 return NULL;
453         }
455         if (!is_merged(sm) || pool_in_pools(sm->pool, keep_stack) || sm_in_keep_
456                 DIMPLIED("kept [stree %d] %s from %d. %s. %s. %s.\n", get_stree_
```



```
457                         is_merged(sm) ? "merged" : "not merged",
458                         pool_in_pools(sm->pool, keep_stack) ? "not in keep pools
459                         sm_in_keep_leafs(sm, keep_stack) ? "reachable keep leaf"
460                 return sm;
461         }
463         DIMPLIED("checking [stree %d] %s from %d (%d) left = %s [stree %d] right
464                  get_stree_id(sm->pool),
465                  show_sm(sm), sm->line, sm->nr_children,
466                  sm->left ? sm->left->state->name : "<none>", sm->left ? get_str
467                  sm->right ? sm->right->state->name : "<none>", sm->right ? get_
468         left = filter_pools(sm->left, remove_stack, keep_stack, &removed, recurs
469         right = filter_pools(sm->right, remove_stack, keep_stack, &removed, recu
470         if (!removed) {
471                 DIMPLIED("kept [stree %d] %s from %d\n", get_stree_id(sm->pool),
472                 return sm;
473         }
474         *modified = 1;
475         if (!left && !right) {
476                 DIMPLIED("removed [stree %d] %s from %d <none>\n", get_stree_id(
477                 return NULL;
478         }
480         if (!left) {
481                 ret = clone_sm(right);
482                 ret->merged = 1;
483                 ret->right = right;
484                 ret->left = NULL;
485         } else if (!right) {
486                 ret = clone_sm(left);
487                 ret->merged = 1;
488                 ret->left = left;
489                 ret->right = NULL;
490         } else {
491                 if (left->sym != sm->sym || strcmp(left->name, sm->name) != 0) {
492                         left = clone_sm(left);
493                         left->sym = sm->sym;
494                         left->name = sm->name;
495                 }
496                 if (right->sym != sm->sym || strcmp(right->name, sm->name) != 0)
497                         right = clone_sm(right);
498                         right->sym = sm->sym;
499                         right->name = sm->name;
500                 }
501                 ret = merge_sm_states(left, right);
502         }
504         ret->pool = sm->pool;
506         DIMPLIED("partial %s => ", show_sm(sm));
507         DIMPLIED("%s from %d [stree %d]\n", show_sm(ret), sm->line, get_stree_id
508         return ret;
509 }
511 static struct stree *filter_stack(struct sm_state *gate_sm,
512                                   struct stree *pre_stree,
513                                   const struct state_list *remove_stack,
514                                   const struct state_list *keep_stack)
515 {
516         struct stree *ret = NULL;
517         struct sm_state *tmp;
518         struct sm_state *filtered_sm;
519         int modified;
520         int recurse_cnt;
521         struct timeval start;
```

```
523         if (!remove_stack)
524                 return NULL;
525
526         if (taking_too_long())
527                 return NULL;
528
529         FOR_EACH_SM(pre_stree, tmp) {
530                 if (option_debug)
531                         sm_msg("%s: %s", __func__, show_sm(tmp));
532                 if (!tmp->merged)
533                         continue;
534                 if (sm_in_keep_leafs(tmp, keep_stack))
535                         continue;
536                 modified = 0;
537                 recurse_cnt = 0;
538                 gettimeofday(&start, NULL);
539                 filtered_sm = filter_pools(tmp, remove_stack, keep_stack, &modif
540                 if (!filtered_sm || !modified)
541                         continue;
542                 /* the assignments here are for borrowed implications */
543                 filtered_sm->name = tmp->name;
544                 filtered_sm->sym = tmp->sym;
545                 avl_insert(&ret, filtered_sm);
546                 if (out_of_memory() || taking_too_long())
547                         return NULL;
548
549         } END_FOR_EACH_SM(tmp);
550         return ret;
551 }
552
553 static void separate_and_filter(struct sm_state *sm, int comparison, struct rang
554                 struct stree *pre_stree,
555                 struct stree **true_states,
556                 struct stree **false_states,
557                 int *mixed)
558 {
559         struct state_list *true_stack = NULL;
560         struct state_list *false_stack = NULL;
561         struct timeval time_before;
562         struct timeval time_after;
563         int sec;
564
565         gettimeofday(&time_before, NULL);
566
567         if (!is_merged(sm)) {
568                 DIMPLIED("%d '%s' is not merged.\n", get_lineno(), sm->name);
569                 return;
570         }
571
572         if (option_debug_implied || option_debug) {
573                 sm_msg("checking implications: (%s %s %s)",
574                        sm->name, show_special(comparison), show_rl(rl));
575         }
576
577         separate_pools(sm, comparison, rl, &true_stack, &false_stack, NULL, mixe
578
579         DIMPLIED("filtering true stack.\n");
580         *true_states = filter_stack(sm, pre_stree, false_stack, true_stack);
581         DIMPLIED("filtering false stack.\n");
582         *false_states = filter_stack(sm, pre_stree, true_stack, false_stack);
583         free_slist(&true_stack);
584         free_slist(&false_stack);
585         if (option_debug_implied || option_debug) {
586                 printf("These are the implied states for the true path: (%s %s %
587                        sm->name, show_special(comparison), show_rl(rl));
588                 __print_stree(*true_states);
```



```
589                 printf("These are the implied states for the false path: (%s %s
590                        sm->name, show_special(comparison), show_rl(rl));
591                 __print_stree(*false_states);
592         }
593
594         gettimeofday(&time_after, NULL);
595         sec = time_after.tv_sec - time_before.tv_sec;
596         if (sec > 20) {
597                 sm->nr_children = 4000;
598                 sm_perror("Function too hairy.  Ignoring implications after %d s
599         }
600 }
601
602 static struct expression *get_last_expr(struct statement *stmt)
603 {
604         struct statement *last;
605
606         last = last_ptr_list((struct ptr_list *)stmt->stmts);
607         if (last->type == STMT_EXPRESSION)
608                 return last->expression;
609
610         if (last->type == STMT_LABEL) {
611                 if (last->label_statement &&
612                     last->label_statement->type == STMT_EXPRESSION)
613                         return last->label_statement->expression;
614         }
615
616         return NULL;
617 }
618
619 static struct expression *get_left_most_expr(struct expression *expr)
620 {
621         struct statement *compound;
622
623         compound = get_expression_statement(expr);
624         if (compound)
625                 return get_last_expr(compound);
626
627         expr = strip_parens(expr);
628         if (expr->type == EXPR_ASSIGNMENT)
629                 return get_left_most_expr(expr->left);
630         return expr;
631 }
632
633 static int is_merged_expr(struct expression  *expr)
634 {
635         struct sm_state *sm;
636         sval_t dummy;
637
638         if (get_value(expr, &dummy))
639                 return 0;
640         sm = get_sm_state_expr(SMATCH_EXTRA, expr);
641         if (!sm)
642                 return 0;
643         if (is_merged(sm))
644                 return 1;
645         return 0;
646 }
647
648 static void delete_gate_sm_equiv(struct stree **stree, const char *name, struct
649 {
650         struct smatch_state *state;
651         struct relation *rel;
652
653         state = get_state(SMATCH_EXTRA, name, sym);
654         if (!state)
```

```
655                 return;
656         FOR_EACH_PTR(estate_related(state), rel) {
657                 delete_state_stree(stree, SMATCH_EXTRA, rel->name, rel->sym);
658         } END_FOR_EACH_PTR(rel);
659 }

661 static void delete_gate_sm(struct stree **stree, const char *name, struct symbol
662 {
663         delete_state_stree(stree, SMATCH_EXTRA, name, sym);
664 }

666 static int handle_comparison(struct expression *expr,
667                              struct stree **implied_true,
668                              struct stree **implied_false)
669 {
670         struct sm_state *sm = NULL;
671         struct range_list *rl = NULL;
672         struct expression *left;
673         struct expression *right;
674         struct symbol *type;
675         int comparison = expr->op;
676         int mixed = 0;

678         left = get_left_most_expr(expr->left);
679         right = get_left_most_expr(expr->right);

681         if (is_merged_expr(left)) {
682                 sm = get_sm_state_expr(SMATCH_EXTRA, left);
683                 get_implied_rl(right, &rl);
684         } else if (is_merged_expr(right)) {
685                 sm = get_sm_state_expr(SMATCH_EXTRA, right);
686                 get_implied_rl(left, &rl);
687                 comparison = flip_comparison(comparison);
688         }

690         if (!rl || !sm)
691                 return 0;

693         type = get_type(expr);
694         if (!type)
695                 return 0;
696         if (type_positive_bits(rl_type(rl)) > type_positive_bits(type))
697                 type = rl_type(rl);
698         if (type_positive_bits(type) < 31)
699                 type = &int_ctype;
700         rl = cast_rl(type, rl);

702         separate_and_filter(sm, comparison, rl, __get_cur_stree(), implied_true,

704         delete_gate_sm_equiv(implied_true, sm->name, sm->sym);
705         delete_gate_sm_equiv(implied_false, sm->name, sm->sym);
706         if (mixed) {
707                 delete_gate_sm(implied_true, sm->name, sm->sym);
708                 delete_gate_sm(implied_false, sm->name, sm->sym);
709         }

711         return 1;
712 }

714 static int handle_zero_comparison(struct expression *expr,
715                                   struct stree **implied_true,
716                                   struct stree **implied_false)
717 {
718         struct symbol *sym;
719         char *name;
720         struct sm_state *sm;
```



```
721         int mixed = 0;
722         int ret = 0;

724         if (expr->type == EXPR_POSTOP)
725                 expr = strip_expr(expr->unop);

727         if (expr->type == EXPR_ASSIGNMENT) {
728                 /* most of the time ->pools will be empty here because we
729                    just set the state, but if have assigned a conditional
730                    function there are implications. */
731                 expr = expr->left;
732         }

734         name = expr_to_var_sym(expr, &sym);
735         if (!name || !sym)
736                 goto free;
737         sm = get_sm_state(SMATCH_EXTRA, name, sym);
738         if (!sm)
739                 goto free;

741         separate_and_filter(sm, SPECIAL_NOTEQUAL, tmp_range_list(estate_type(sm-
742         delete_gate_sm_equiv(implied_true, sm->name, sm->sym);
743         delete_gate_sm_equiv(implied_false, sm->name, sm->sym);
744         if (mixed) {
745                 delete_gate_sm(implied_true, sm->name, sm->sym);
746                 delete_gate_sm(implied_false, sm->name, sm->sym);
747         }

749         ret = 1;
750 free:
751         free_string(name);
752         return ret;
753 }

755 static int handled_by_comparison_hook(struct expression *expr,
756                                       struct stree **implied_true,
757                                       struct stree **implied_false)
758 {
759         struct state_list *true_stack = NULL;
760         struct state_list *false_stack = NULL;
761         struct stree *pre_stree;
762         struct sm_state *sm;

764         sm = comparison_implication_hook(expr, &true_stack, &false_stack);
765         if (!sm)
766                 return 0;

768         pre_stree = clone_stree(__get_cur_stree());

770         *implied_true = filter_stack(sm, pre_stree, false_stack, true_stack);
771         *implied_false = filter_stack(sm, pre_stree, true_stack, false_stack);

773         free_stree(&pre_stree);
774         free_slist(&true_stack);
775         free_slist(&false_stack);

777         return 1;
778 }

780 static int handled_by_extra_states(struct expression *expr,
781                                    struct stree **implied_true,
782                                    struct stree **implied_false)
783 {
784         if (expr->type == EXPR_COMPARE)
785                 return handle_comparison(expr, implied_true, implied_false);
786         else
```

```
787                 return handle_zero_comparison(expr, implied_true, implied_false)
788 }

790 static int handled_by_stored_conditions(struct expression *expr,
791                                         struct stree **implied_true,
792                                         struct stree **implied_false)
793 {
794         struct state_list *true_stack = NULL;
795         struct state_list *false_stack = NULL;
796         struct stree *pre_stree;
797         struct sm_state *sm;

799         sm = stored_condition_implication_hook(expr, &true_stack, &false_stack);
800         if (!sm)
801                 return 0;

803         pre_stree = clone_stree(__get_cur_stree());

805         *implied_true = filter_stack(sm, pre_stree, false_stack, true_stack);
806         *implied_false = filter_stack(sm, pre_stree, true_stack, false_stack);

808         free_stree(&pre_stree);
809         free_slist(&true_stack);
810         free_slist(&false_stack);

812         return 1;
813 }

815 static int found_implications;
816 static struct stree *saved_implied_true;
817 static struct stree *saved_implied_false;
818 static struct stree *extra_saved_implied_true;
819 static struct stree *extra_saved_implied_false;

821 static void separate_extra_states(struct stree **implied_true,
822                                   struct stree **implied_false)
823 {
824         struct sm_state *sm;

826         /* We process extra states later to preserve the implications. */
827         FOR_EACH_SM(*implied_true, sm) {
828                 if (sm->owner == SMATCH_EXTRA)
829                         overwrite_sm_state_stree(&extra_saved_implied_true, sm);
830         } END_FOR_EACH_SM(sm);
831         FOR_EACH_SM(extra_saved_implied_true, sm) {
832                 delete_state_stree(implied_true, sm->owner, sm->name, sm->sym);
833         } END_FOR_EACH_SM(sm);

835         FOR_EACH_SM(*implied_false, sm) {
836                 if (sm->owner == SMATCH_EXTRA)
837                         overwrite_sm_state_stree(&extra_saved_implied_false, sm)
838         } END_FOR_EACH_SM(sm);
839         FOR_EACH_SM(extra_saved_implied_false, sm) {
840                 delete_state_stree(implied_false, sm->owner, sm->name, sm->sym);
841         } END_FOR_EACH_SM(sm);
842 }

844 static void get_tf_states(struct expression *expr,
845                           struct stree **implied_true,
846                           struct stree **implied_false)
847 {
848         if (handled_by_comparison_hook(expr, implied_true, implied_false))
849                 goto found;

851         if (handled_by_extra_states(expr, implied_true, implied_false)) {
852                 separate_extra_states(implied_true, implied_false);
```



```
853                 goto found;
854         }

856         if (handled_by_stored_conditions(expr, implied_true, implied_false))
857                 goto found;

859         return;
860 found:
861         found_implications = 1;
862 }

864 static void save_implications_hook(struct expression *expr)
865 {
866         if (taking_too_long())
867                 return;
868         get_tf_states(expr, &saved_implied_true, &saved_implied_false);
869 }

871 static void set_implied_states(struct expression *expr)
872 {
873         struct sm_state *sm;

875         FOR_EACH_SM(saved_implied_true, sm) {
876                 __set_true_false_sm(sm, NULL);
877         } END_FOR_EACH_SM(sm);
878         free_stree(&saved_implied_true);

880         FOR_EACH_SM(saved_implied_false, sm) {
881                 __set_true_false_sm(NULL, sm);
882         } END_FOR_EACH_SM(sm);
883         free_stree(&saved_implied_false);
884 }

886 static void set_extra_implied_states(struct expression *expr)
887 {
888         saved_implied_true = extra_saved_implied_true;
889         saved_implied_false = extra_saved_implied_false;
890         extra_saved_implied_true = NULL;
891         extra_saved_implied_false = NULL;
892         set_implied_states(NULL);
893 }

895 void param_limit_implications(struct expression *expr, int param, char *key, cha
896 {
897         struct expression *arg;
898         struct symbol *compare_type;
899         char *name;
900         struct symbol *sym;
901         struct sm_state *sm;
902         struct sm_state *tmp;
903         struct stree *implied_true = NULL;
904         struct stree *implied_false = NULL;
905         struct range_list *orig, *limit;

907         while (expr->type == EXPR_ASSIGNMENT)
908                 expr = strip_expr(expr->right);
909         if (expr->type != EXPR_CALL)
910                 return;

912         arg = get_argument_from_call_expr(expr->args, param);
913         if (!arg)
914                 return;

916         arg = strip_parens(arg);
917         while (arg->type == EXPR_ASSIGNMENT && arg->op == '=')
918                 arg = strip_parens(arg->left);
```

```
920         name = get_variable_from_key(arg, key, &sym);
921         if (!name || !sym)
922                 goto free;
923
924         sm = get_sm_state(SMATCH_EXTRA, name, sym);
925         if (!sm || !sm->merged)
926                 goto free;
927
928         if (strcmp(key, "$") == 0)
929                 compare_type = get_arg_type(expr->fn, param);
930         else
931                 compare_type = get_member_type_from_key(arg, key);
932
933         orig = estate_rl(sm->state);
934         orig = cast_rl(compare_type, orig);
935
936         call_results_to_rl(expr, compare_type, value, &limit);
937
938         separate_and_filter(sm, SPECIAL_EQUAL, limit, __get_cur_stree(), &implie
939
940         FOR_EACH_SM(implied_true, tmp) {
941                 __set_sm_fake_stree(tmp);
942         } END_FOR_EACH_SM(tmp);
943
944         free_stree(&implied_true);
945         free_stree(&implied_false);
946 free:
947         free_string(name);
948 }
949
950 struct stree *__implied_case_stree(struct expression *switch_expr,
951                                    struct range_list *rl,
952                                    struct range_list_stack **remaining_cases,
953                                    struct stree **raw_stree)
954 {
955         char *name;
956         struct symbol *sym;
957         struct var_sym_list *vsl;
958         struct sm_state *sm;
959         struct stree *true_states = NULL;
960         struct stree *false_states = NULL;
961         struct stree *extra_states;
962         struct stree *ret = clone_stree(*raw_stree);
963
964         name = expr_to_chunk_sym_vsl(switch_expr, &sym, &vsl);
965
966         if (rl)
967                 filter_top_rl(remaining_cases, rl);
968         else
969                 rl = clone_rl(top_rl(*remaining_cases));
970
971         if (name) {
972                 sm = get_sm_state_stree(*raw_stree, SMATCH_EXTRA, name, sym);
973                 if (sm)
974                         separate_and_filter(sm, SPECIAL_EQUAL, rl, *raw_stree, &
975         }
976
977         __push_fake_cur_stree();
978         __unnullify_path();
979         if (name)
980                 set_extra_nomod_vsl(name, sym, vsl, NULL, alloc_estate_rl(rl));
981         __pass_case_to_client(switch_expr, rl);
982         extra_states = __pop_fake_cur_stree();
983         overwrite_stree(extra_states, &true_states);
984         overwrite_stree(true_states, &ret);
```



```
985         free_stree(&extra_states);
986         free_stree(&true_states);
987         free_stree(&false_states);
988
989         free_string(name);
990         return ret;
991 }
992
993 static void match_end_func(struct symbol *sym)
994 {
995         if (__inline_fn)
996                 return;
997         implied_debug_msg = NULL;
998 }
999
1000 static void get_tf_stacks_from_pool(struct sm_state *gate_sm,
1001                                     struct sm_state *pool_sm,
1002                                     struct state_list **true_stack,
1003                                     struct state_list **false_stack)
1004 {
1005         struct sm_state *tmp;
1006         int possibly_true = 0;
1007
1008         if (!gate_sm)
1009                 return;
1010
1011         if (strcmp(gate_sm->state->name, pool_sm->state->name) == 0) {
1012                 add_ptr_list(true_stack, pool_sm);
1013                 return;
1014         }
1015
1016         FOR_EACH_PTR(gate_sm->possible, tmp) {
1017                 if (strcmp(tmp->state->name, pool_sm->state->name) == 0) {
1018                         possibly_true = 1;
1019                         break;
1020                 }
1021         } END_FOR_EACH_PTR(tmp);
1022
1023         if (!possibly_true) {
1024                 add_ptr_list(false_stack, gate_sm);
1025                 return;
1026         }
1027
1028         get_tf_stacks_from_pool(gate_sm->left, pool_sm, true_stack, false_stack)
1029         get_tf_stacks_from_pool(gate_sm->right, pool_sm, true_stack, false_stack
1030 }
1031
1032 /*
1033  * The situation is we have a SMATCH_EXTRA state and we want to break it into
1034  * each of the ->possible states and find the implications of each.  The caller
1035  * has to use __push_fake_cur_stree() to preserve the correct states so they
1036  * can be restored later.
1037  */
1038 void overwrite_states_using_pool(struct sm_state *gate_sm, struct sm_state *pool
1039 {
1040         struct state_list *true_stack = NULL;
1041         struct state_list *false_stack = NULL;
1042         struct stree *pre_stree;
1043         struct stree *implied_true;
1044         struct sm_state *tmp;
1045
1046         if (!pool_sm->pool)
1047                 return;
1048
1049         get_tf_stacks_from_pool(gate_sm, pool_sm, &true_stack, &false_stack);
```

```
1051         pre_stree = clone_stree(__get_cur_stree());

1053         implied_true = filter_stack(gate_sm, pre_stree, false_stack, true_stack)

1055         free_stree(&pre_stree);
1056         free_slist(&true_stack);
1057         free_slist(&false_stack);

1059         FOR_EACH_SM(implied_true, tmp) {
1060                 set_state(tmp->owner, tmp->name, tmp->sym, tmp->state);
1061         } END_FOR_EACH_SM(tmp);

1063         free_stree(&implied_true);
1064 }

1066 int assume(struct expression *expr)
1067 {
1068         int orig_final_pass = final_pass;

1070         in_fake_env++;
1071         final_pass = 0;
1072         __push_fake_cur_stree();
1073         found_implications = 0;
1074         __split_whole_condition(expr);
1075         final_pass = orig_final_pass;
1076         in_fake_env--;

1078         return 1;
1079 }

1081 void end_assume(void)
1082 {
1083         __discard_false_states();
1084         __free_fake_cur_stree();
1085 }

1087 int impossible_assumption(struct expression *left, int op, sval_t sval)
1088 {
1089         struct expression *value;
1090         struct expression *comparison;
1091         int ret;

1093         value = value_expr(sval.value);
1094         comparison = compare_expression(left, op, value);

1096         if (!assume(comparison))
1097                 return 0;
1098         ret = is_impossible_path();
1099         end_assume();

1101         return ret;
1102 }

1104 void __extra_match_condition(struct expression *expr);
1105 void __comparison_match_condition(struct expression *expr);
1106 void __stored_condition(struct expression *expr);
1107 void register_implications(int id)
1108 {
1109         add_hook(&save_implications_hook, CONDITION_HOOK);
1110         add_hook(&set_implied_states, CONDITION_HOOK);
1111         add_hook(&__extra_match_condition, CONDITION_HOOK);
1112         add_hook(&set_extra_implied_states, CONDITION_HOOK);
1113         add_hook(&__comparison_match_condition, CONDITION_HOOK);
1114         add_hook(&__stored_condition, CONDITION_HOOK);
1115         add_hook(&match_end_func, END_FUNC_HOOK);
1116 }
```

```
**********************************************************
    2796 Fri Dec 21 15:00:26 2018
new/usr/src/tools/smatch/src/smatch_impossible.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int my_id;
static int my_return_id;

STATE(impossible);

int is_impossible_path(void)
{
        if (get_state(my_id, "impossible", NULL) == &impossible)
                return 1;
        return 0;
}

static void handle_compare(struct expression *left, int op, struct expression *r
{
        int true_impossible = 0;
        int false_impossible = 0;

        left = strip_expr(left);
        while (left && left->type == EXPR_ASSIGNMENT)
                left = strip_expr(left->left);

        if (!possibly_true(left, op, right))
                true_impossible = 1;
        if (!possibly_false(left, op, right))
                false_impossible = 1;

        if (!true_impossible && !false_impossible)
                return;

        set_true_false_states(my_id, "impossible", NULL,
                              true_impossible ? &impossible : NULL,
                              false_impossible ? &impossible : NULL);

        if (inside_loop())
                return;

        set_true_false_states(my_return_id, "impossible", NULL,
                              true_impossible ? &impossible : NULL,
                              false_impossible ? &impossible : NULL);
}
```



```
static void match_condition(struct expression *expr)
{
        if (expr->type == EXPR_COMPARE)
                handle_compare(expr->left, expr->op, expr->right);
        else
                handle_compare(expr, SPECIAL_NOTEQUAL, zero_expr());
}

void set_path_impossible(void)
{
        set_state(my_id, "impossible", NULL, &impossible);

        if (inside_loop())
                return;

        set_state(my_return_id, "impossible", NULL, &impossible);
}

static void match_case(struct expression *expr, struct range_list *rl)
{
        if (rl)
                return;
        set_path_impossible();
}

static void print_impossible_return(int return_id, char *return_ranges, struct e
{
        if (get_state(my_return_id, "impossible", NULL) == &impossible) {
                if (option_debug)
                        sm_msg("impossible return.  return_id = %d return ranges
                sql_insert_return_states(return_id, return_ranges, CULL_PATH, -1
        }
}

void register_impossible(int id)
{
        my_id = id;

        add_hook(&match_condition, CONDITION_HOOK);
        add_hook(&match_case, CASE_HOOK);
}

void register_impossible_return(int id)
{
        my_return_id = id;

        add_split_return_callback(&print_impossible_return);
}
```

```
**********************************************************
   30546 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_kernel_user_data.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * There are a couple checks that try to see if a variable
 * comes from the user.  It would be better to unify them
 * into one place.  Also it we should follow the data down
 * the call paths.  Hence this file.
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;
static int my_call_id;

STATE(called);
static bool func_gets_user_data;

static const char * kstr_funcs[] = {
        "kstrtoull", "kstrtoll", "kstrtoul", "kstrtol", "kstrtouint",
        "kstrtoint", "kstrtou64", "kstrtos64", "kstrtou32", "kstrtos32",
        "kstrtou16", "kstrtos16", "kstrtou8", "kstrtos8", "kstrtoull_from_user"
        "kstrtoll_from_user", "kstrtoul_from_user", "kstrtol_from_user",
        "kstrtouint_from_user", "kstrtoint_from_user", "kstrtou16_from_user",
        "kstrtos16_from_user", "kstrtou8_from_user", "kstrtos8_from_user",
        "kstrtou64_from_user", "kstrtos64_from_user", "kstrtou32_from_user",
        "kstrtos32_from_user",
};

static const char *returns_user_data[] = {
        "simple_strtol", "simple_strtoll", "simple_strtoul", "simple_strtoull",
        "kvm_register_read", "nlmsg_data", "nla_data", "memdup_user",
        "kmap_atomic", "skb_network_header",
};

static void set_points_to_user_data(struct expression *expr);

static struct stree *start_states;
static struct stree_stack *saved_stack;
static void save_start_states(struct statement *stmt)
{
        start_states = clone_stree(__get_cur_stree());
}
```



```
static void free_start_states(void)
{
        free_stree(&start_states);
}

static void match_save_states(struct expression *expr)
{
        push_stree(&saved_stack, start_states);
        start_states = NULL;
}

static void match_restore_states(struct expression *expr)
{
        free_stree(&start_states);
        start_states = pop_stree(&saved_stack);
}

static struct smatch_state *empty_state(struct sm_state *sm)
{
        return alloc_estate_empty();
}

static void pre_merge_hook(struct sm_state *sm)
{
        struct smatch_state *user;
        struct smatch_state *extra;
        struct range_list *rl;
        sval_t dummy;
        sval_t sval_100;

        sval_100.value = 100;
        sval_100.type = &int_ctype;

        user = get_state(my_id, sm->name, sm->sym);
        if (!user)
                return;
        if (!__in_function_def && !estate_rl(sm->state)) {
                /*
                 * If the one side is capped and the other side is empty then
                 * let's just mark it as not-user data because the information
                 * isn't going to be useful.  How this looks is:
                 *
                 * if (user_var > trusted)
                 *      user_var = trusted;  <-- empty state
                 * else
                 *      <-- capped
                 *
                 * The problem is that sometimes things are capped to a literal
                 * and we'd like to keep the state in that case...  Ugh.  I've
                 * added a check which assumes that everything less than 100 is
                 * probably capped against a literal.
                 *
                 */
                if (is_capped_var_sym(sm->name, sm->sym) &&
                    sval_cmp(estate_max(user), sval_100) > 0)
                        set_state(my_id, sm->name, sm->sym, alloc_estate_empty()
                return;
        }
        extra = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (!extra || !estate_rl(extra))
                return;
        rl = rl_intersection(estate_rl(user), estate_rl(extra));
        if (rl_to_sval(rl, &dummy))
                rl = NULL;
        set_state(my_id, sm->name, sm->sym, alloc_estate_rl(clone_rl(rl)));
}
```

```
128 static void extra_nomod_hook(const char *name, struct symbol *sym, struct expres
129 {
130         struct smatch_state *user;
131         struct range_list *rl;
132
133         user = get_state(my_id, name, sym);
134         if (!user)
135                 return;
136         rl = rl_intersection(estate_rl(user), estate_rl(state));
137         if (rl_equiv(rl, estate_rl(user)))
138                 return;
139         set_state(my_id, name, sym, alloc_estate_rl(rl));
140 }
141
142 static void tag_inner_struct_members(struct expression *expr, struct symbol *mem
143 {
144         struct expression *edge_member;
145         struct symbol *base = get_real_base_type(member);
146         struct symbol *tmp;
147
148         if (member->ident)
149                 expr = member_expression(expr, '.', member->ident);
150
151         FOR_EACH_PTR(base->symbol_list, tmp) {
152                 struct symbol *type;
153
154                 type = get_real_base_type(tmp);
155                 if (!type)
156                         continue;
157
158                 if (type->type == SYM_UNION || type->type == SYM_STRUCT) {
159                         tag_inner_struct_members(expr, tmp);
160                         continue;
161                 }
162
163                 if (!tmp->ident)
164                         continue;
165
166                 edge_member = member_expression(expr, '.', tmp->ident);
167                 set_state_expr(my_id, edge_member, alloc_estate_whole(type));
168         } END_FOR_EACH_PTR(tmp);
169 }
170
171 static void tag_struct_members(struct symbol *type, struct expression *expr)
172 {
173         struct symbol *tmp;
174         struct expression *member;
175         int op = '*';
176
177         if (expr->type == EXPR_PREOP && expr->op == '&') {
178                 expr = strip_expr(expr->unop);
179                 op = '.';
180         }
181
182         FOR_EACH_PTR(type->symbol_list, tmp) {
183                 type = get_real_base_type(tmp);
184                 if (!type)
185                         continue;
186
187                 if (type->type == SYM_UNION || type->type == SYM_STRUCT) {
188                         tag_inner_struct_members(expr, tmp);
189                         continue;
190                 }
191
192                 if (!tmp->ident)
```



```
193                         continue;
194
195                 member = member_expression(expr, op, tmp->ident);
196                 set_state_expr(my_id, member, alloc_estate_whole(get_type(member
197
198                 if (type->type == SYM_ARRAY)
199                         set_points_to_user_data(member);
200         } END_FOR_EACH_PTR(tmp);
201 }
202
203 static void tag_base_type(struct expression *expr)
204 {
205         if (expr->type == EXPR_PREOP && expr->op == '&')
206                 expr = strip_expr(expr->unop);
207         else
208                 expr = deref_expression(expr);
209         set_state_expr(my_id, expr, alloc_estate_whole(get_type(expr)));
210 }
211
212 static void tag_as_user_data(struct expression *expr)
213 {
214         struct symbol *type;
215
216         expr = strip_expr(expr);
217
218         type = get_type(expr);
219         if (!type || type->type != SYM_PTR)
220                 return;
221         type = get_real_base_type(type);
222         if (!type)
223                 return;
224         if (type == &void_ctype) {
225                 set_state_expr(my_id, deref_expression(expr), alloc_estate_whole
226                 return;
227         }
228         if (type->type == SYM_BASETYPE)
229                 tag_base_type(expr);
230         if (type->type == SYM_STRUCT || type->type == SYM_UNION) {
231                 if (expr->type != EXPR_PREOP || expr->op != '&')
232                         expr = deref_expression(expr);
233                 else
234                         set_state_expr(my_id, deref_expression(expr), alloc_esta
235                 tag_struct_members(type, expr);
236         }
237 }
238
239 static void match_user_copy(const char *fn, struct expression *expr, void *_para
240 {
241         int param = PTR_INT(_param);
242         struct expression *dest;
243
244         func_gets_user_data = true;
245
246         dest = get_argument_from_call_expr(expr->args, param);
247         dest = strip_expr(dest);
248         if (!dest)
249                 return;
250         tag_as_user_data(dest);
251 }
252
253 static int is_dev_attr_name(struct expression *expr)
254 {
255         char *name;
256         int ret = 0;
257
258         name = expr_to_str(expr);
```

```
259         if (!name)
260                 return 0;
261         if (strstr(name, "->attr.name"))
262                 ret = 1;
263         free_string(name);
264         return ret;
265 }

267 static int ends_in_n(struct expression *expr)
268 {
269         struct string *str;

271         if (!expr)
272                 return 0;
273         if (expr->type != EXPR_STRING || !expr->string)
274                 return 0;

276         str = expr->string;
277         if (str->length < 3)
278                 return 0;

280         if (str->data[str->length - 3] == '%' &&
281             str->data[str->length - 2] == 'n')
282                 return 1;
283         return 0;
284 }

286 static void match_sscanf(const char *fn, struct expression *expr, void *unused)
287 {
288         struct expression *str, *format, *arg;
289         int i, last;

291         func_gets_user_data = true;

293         str = get_argument_from_call_expr(expr->args, 0);
294         if (is_dev_attr_name(str))
295                 return;

297         format = get_argument_from_call_expr(expr->args, 1);
298         if (is_dev_attr_name(format))
299                 return;

301         last = ptr_list_size((struct ptr_list *)expr->args) - 1;

303         i = -1;
304         FOR_EACH_PTR(expr->args, arg) {
305                 i++;
306                 if (i < 2)
307                         continue;
308                 if (i == last && ends_in_n(format))
309                         continue;
310                 tag_as_user_data(arg);
311         } END_FOR_EACH_PTR(arg);
312 }

314 static int is_skb_data(struct expression *expr)
315 {
316         struct symbol *sym;

318         if (!expr)
319                 return 0;

321         if (expr->type == EXPR_BINOP && expr->op == '+')
322                 return is_skb_data(expr->left);

324         expr = strip_expr(expr);
```



```
325         if (!expr)
326                 return 0;
327         if (expr->type != EXPR_DEREF || expr->op != '.')
328                 return 0;

330         if (!expr->member)
331                 return 0;
332         if (strcmp(expr->member->name, "data") != 0)
333                 return 0;

335         sym = expr_to_sym(expr->deref);
336         if (!sym)
337                 return 0;
338         sym = get_real_base_type(sym);
339         if (!sym || sym->type != SYM_PTR)
340                 return 0;
341         sym = get_real_base_type(sym);
342         if (!sym || sym->type != SYM_STRUCT || !sym->ident)
343                 return 0;
344         if (strcmp(sym->ident->name, "sk_buff") != 0)
345                 return 0;

347         return 1;
348 }

350 static int get_rl_from_function(struct expression *expr, struct range_list **rl)
351 {
352         int i;

354         if (expr->type != EXPR_CALL || expr->fn->type != EXPR_SYMBOL ||
355             !expr->fn->symbol_name || !expr->fn->symbol_name->name)
356                 return 0;

358         for (i = 0; i < ARRAY_SIZE(returns_user_data); i++) {
359                 if (strcmp(expr->fn->symbol_name->name, returns_user_data[i]) ==
360                         *rl = alloc_whole_rl(get_type(expr));
361                         return 1;
362                 }
363         }
364         return 0;
365 }

367 int points_to_user_data(struct expression *expr)
368 {
369         struct smatch_state *state;
370         struct range_list *rl;
371         char buf[256];
372         struct symbol *sym;
373         char *name;
374         int ret = 0;

376         expr = strip_expr(expr);
377         if (!expr)
378                 return 0;
379         if (is_skb_data(expr))
380                 return 1;
381         if (get_rl_from_function(expr, &rl))
382                 return 1;

384         if (expr->type == EXPR_BINOP && expr->op == '+') {
385                 if (points_to_user_data(expr->left))
386                         return 1;
387                 if (points_to_user_data(expr->right))
388                         return 1;
389                 return 0;
390         }
```

```
392         name = expr_to_var_sym(expr, &sym);
393         if (!name || !sym)
394                 goto free;
395         snprintf(buf, sizeof(buf), "*%s", name);
396         state = get_state(my_id, buf, sym);
397         if (state && estate_rl(state))
398                 ret = 1;
399 free:
400         free_string(name);
401         return ret;
402 }

404 static void set_points_to_user_data(struct expression *expr)
405 {
406         char *name;
407         struct symbol *sym;
408         char buf[256];

410         name = expr_to_var_sym(expr, &sym);
411         if (!name || !sym)
412                 goto free;
413         snprintf(buf, sizeof(buf), "*%s", name);
414         set_state(my_id, buf, sym, alloc_estate_whole(&llong_ctype));
415 free:
416         free_string(name);
417 }

419 static int comes_from_skb_data(struct expression *expr)
420 {
421         expr = strip_expr(expr);
422         if (!expr || expr->type != EXPR_PREOP || expr->op != '*')
423                 return 0;

425         expr = strip_expr(expr->unop);
426         if (!expr)
427                 return 0;
428         if (expr->type == EXPR_BINOP && expr->op == '+')
429                 expr = strip_expr(expr->left);

431         return is_skb_data(expr);
432 }

434 static int handle_struct_assignment(struct expression *expr)
435 {
436         struct expression *right;
437         struct symbol *left_type, *right_type;

439         left_type = get_type(expr->left);
440         if (!left_type || left_type->type != SYM_PTR)
441                 return 0;
442         left_type = get_real_base_type(left_type);
443         if (!left_type)
444                 return 0;
445         if (left_type->type != SYM_STRUCT &&
446             left_type->type != SYM_UNION)
447                 return 0;

449         /*
450          * Ignore struct to struct assignments because for those we look at the
451          * individual members.
452          */
453         right = strip_expr(expr->right);
454         right_type = get_type(right);
455         if (!right_type || right_type->type != SYM_PTR)
456                 return 0;
```



```
458         /* If we are assigning struct members then normally that is handled
459          * by fake assignments, however if we cast one struct to a different
460          * of struct then we handle that here.
461          */
462         right_type = get_real_base_type(right_type);
463         if (right_type == left_type)
464                 return 0;

466         if (!points_to_user_data(right))
467                 return 0;

469         tag_as_user_data(expr->left);
470         return 1;
471 }

473 static int handle_get_user(struct expression *expr)
474 {
475         char *name;
476         int ret = 0;

478         name = get_macro_name(expr->pos);
479         if (!name || strcmp(name, "get_user") != 0)
480                 return 0;

482         name = expr_to_var(expr->right);
483         if (!name || strcmp(name, "__val_gu") != 0)
484                 goto free;
485         set_state_expr(my_id, expr->left, alloc_estate_whole(get_type(expr->left
486         ret = 1;
487 free:
488         free_string(name);
489         return ret;
490 }

492 static void match_assign(struct expression *expr)
493 {
494         struct range_list *rl;

496         if (is_fake_call(expr->right))
497                 goto clear_old_state;
498         if (handle_get_user(expr))
499                 return;
500         if (points_to_user_data(expr->right))
501                 set_points_to_user_data(expr->left);
502         if (handle_struct_assignment(expr))
503                 return;

505         if (!get_user_rl(expr->right, &rl))
506                 goto clear_old_state;

508         rl = cast_rl(get_type(expr->left), rl);
509         set_state_expr(my_id, expr->left, alloc_estate_rl(rl));

511         return;

513 clear_old_state:
514         if (get_state_expr(my_id, expr->left))
515                 set_state_expr(my_id, expr->left, alloc_estate_empty());
516 }

518 static void handle_eq_noteq(struct expression *expr)
519 {
520         struct smatch_state *left_orig, *right_orig;

522         left_orig = get_state_expr(my_id, expr->left);
```

```
523         right_orig = get_state_expr(my_id, expr->right);

525         if (!left_orig && !right_orig)
526                 return;
527         if (left_orig && right_orig)
528                 return;

530         if (left_orig) {
531                 set_true_false_states_expr(my_id, expr->left,
532                                 expr->op == SPECIAL_EQUAL ? alloc_estate_empty()
533                                 expr->op == SPECIAL_EQUAL ? NULL : alloc_estate_
534         } else {
535                 set_true_false_states_expr(my_id, expr->right,
536                                 expr->op == SPECIAL_EQUAL ? alloc_estate_empty()
537                                 expr->op == SPECIAL_EQUAL ? NULL : alloc_estate_
538         }
539 }

541 static void handle_unsigned_lt_gt(struct expression *expr)
542 {
543         struct symbol *type;
544         struct range_list *left;
545         struct range_list *right;
546         struct range_list *non_negative;
547         sval_t min, minus_one;

549         /*
550          * conditions are mostly handled by smatch_extra.c.  The special case
551          * here is that say you have if (user_int < unknown_u32) {
552          * In Smatch extra we say that, We have no idea what value
553          * unknown_u32 is so the only thin we can say for sure is that
554          * user_int is not -1 (UINT_MAX).  But in check_user_data2.c we should
555          * assume that unless unknown_u32 is user data, it's probably less than
556          * INT_MAX.
557          *
558          */

560         type = get_type(expr);
561         if (!type_unsigned(type))
562                 return;

564         /*
565          * Assume if (user < trusted) { ... because I am lazy and because this
566          * is the correct way to write code.
567          */
568         if (!get_user_rl(expr->left, &left))
569                 return;
570         if (get_user_rl(expr->right, &right))
571                 return;

573         if (!sval_is_negative(rl_min(left)))
574                 return;
575         min = rl_min(left);
576         minus_one.type = rl_type(left);
577         minus_one.value = -1;
578         non_negative = remove_range(left, min, minus_one);

580         switch (expr->op) {
581         case '<':
582         case SPECIAL_UNSIGNED_LT:
583         case SPECIAL_LTE:
584         case SPECIAL_UNSIGNED_LTE:
585                 set_true_false_states_expr(my_id, expr->left,
586                                            alloc_estate_rl(non_negative), NULL);
587                 break;
588         case '>':
```



```
589         case SPECIAL_UNSIGNED_GT:
590         case SPECIAL_GTE:
591         case SPECIAL_UNSIGNED_GTE:
592                 set_true_false_states_expr(my_id, expr->left,
593                                            NULL, alloc_estate_rl(non_negative));
594                 break;
595         }
596 }

598 static void match_condition(struct expression *expr)
599 {
600         if (expr->type != EXPR_COMPARE)
601                 return;

603         if (expr->op == SPECIAL_EQUAL ||
604             expr->op == SPECIAL_NOTEQUAL) {
605                 handle_eq_noteq(expr);
606                 return;
607         }

609         handle_unsigned_lt_gt(expr);
610 }

612 static void match_user_assign_function(const char *fn, struct expression *expr,
613 {
614         tag_as_user_data(expr->left);
615         set_points_to_user_data(expr->left);
616 }

618 static void match_returns_user_rl(const char *fn, struct expression *expr, void
619 {
620         func_gets_user_data = true;
621 }

623 static int get_user_macro_rl(struct expression *expr, struct range_list **rl)
624 {
625         struct expression *parent;
626         char *macro;

628         if (!expr)
629                 return 0;

631         macro = get_macro_name(expr->pos);
632         if (!macro)
633                 return 0;

635         /* handle ntohl(foo[i]) where "i" is trusted */
636         parent = expr_get_parent_expr(expr);
637         while (parent && parent->type != EXPR_BINOP)
638                 parent = expr_get_parent_expr(parent);
639         if (parent && parent->type == EXPR_BINOP) {
640                 char *parent_macro = get_macro_name(parent->pos);

642                 if (parent_macro && strcmp(macro, parent_macro) == 0)
643                         return 0;
644         }

646         if (strcmp(macro, "ntohl") == 0) {
647                 *rl = alloc_whole_rl(&uint_ctype);
648                 return 1;
649         }
650         if (strcmp(macro, "ntohs") == 0) {
651                 *rl = alloc_whole_rl(&ushort_ctype);
652                 return 1;
653         }
654         return 0;
```

```
655 }
657 struct db_info {
658         struct range_list *rl;
659         struct expression *call;
660 };
661 static int returned_rl_callback(void *_info, int argc, char **argv, char **azCol
662 {
663         struct db_info *db_info = _info;
664         struct range_list *rl;
665         char *return_ranges = argv[0];
666         char *user_ranges = argv[1];
667         struct expression *arg;
668         int comparison;
670         if (argc != 2)
671                 return 0;
673         call_results_to_rl(db_info->call, get_type(db_info->call), user_ranges,
674         if (str_to_comparison_arg(return_ranges, db_info->call, &comparison, &ar
675             comparison == SPECIAL_EQUAL) {
676                 struct range_list *orig_rl;
678                 if (!get_user_rl(arg, &orig_rl))
679                         return 0;
680                 rl = rl_intersection(rl, orig_rl);
681                 if (!rl)
682                         return 0;
683         }
684         db_info->rl = rl_union(db_info->rl, rl);
686         return 0;
687 }
689 static int has_user_data(struct symbol *sym)
690 {
691         struct sm_state *tmp;
693         FOR_EACH_MY_SM(my_id, __get_cur_stree(), tmp) {
694                 if (tmp->sym == sym)
695                         return 1;
696         } END_FOR_EACH_SM(tmp);
697         return 0;
698 }
700 static int we_pass_user_data(struct expression *call)
701 {
702         struct expression *arg;
703         struct symbol *sym;
705         FOR_EACH_PTR(call->args, arg) {
706                 sym = expr_to_sym(arg);
707                 if (!sym)
708                         continue;
709                 if (has_user_data(sym))
710                         return 1;
711         } END_FOR_EACH_PTR(arg);
713         return 0;
714 }
716 static int db_returned_user_rl(struct expression *call, struct range_list **rl)
717 {
718         struct db_info db_info = {};
720         /* for function pointers assume everything is used */
```



```
721         if (call->fn->type != EXPR_SYMBOL)
722                 return 0;
723         if (is_fake_call(call))
724                 return 0;
726         db_info.call = call;
727         run_sql(&returned_rl_callback, &db_info,
728                 "select return, value from return_states where %s and type = %d
729                 get_static_filter(call->fn->symbol), USER_DATA3_SET);
730         if (db_info.rl) {
731                 func_gets_user_data = true;
732                 *rl = db_info.rl;
733                 return 1;
734         }
736         run_sql(&returned_rl_callback, &db_info,
737                 "select return, value from return_states where %s and type = %d
738                 get_static_filter(call->fn->symbol), USER_DATA3);
739         if (db_info.rl) {
740                 if (!we_pass_user_data(call))
741                         return 0;
742                 *rl = db_info.rl;
743                 return 1;
744         }
746         return 0;
747 }
749 struct stree *get_user_stree(void)
750 {
751         return get_all_states_stree(my_id);
752 }
754 static int user_data_flag;
755 static int no_user_data_flag;
756 static struct range_list *var_user_rl(struct expression *expr)
757 {
758         struct smatch_state *state;
759         struct range_list *rl;
760         struct range_list *absolute_rl;
762         if (expr->type == EXPR_BINOP && expr->op == '%') {
763                 struct range_list *left, *right;
765                 if (!get_user_rl(expr->right, &right))
766                         return NULL;
767                 get_absolute_rl(expr->left, &left);
768                 rl = rl_binop(left, '%', right);
769                 goto found;
770         }
772         if (!option_spammy && expr->type == EXPR_BINOP && expr->op == '/') {
773                 struct range_list *left = NULL;
774                 struct range_list *right = NULL;
775                 struct range_list *abs_right;
777                 /*
778                  * The specific bug I'm dealing with is:
779                  *
780                  * foo = capped_user / unknown;
781                  *
782                  * Instead of just saying foo is now entirely user_rl we should
783                  * probably say instead that it is not at all user data.
784                  *
785                  */
```

```
787                 get_user_rl(expr->left, &left);
788                 get_user_rl(expr->right, &right);
789                 get_absolute_rl(expr->right, &abs_right);

791                 if (left && !right) {
792                         rl = rl_binop(left, '/', abs_right);
793                         if (sval_cmp(rl_max(left), rl_max(rl)) < 0)
794                                 no_user_data_flag = 1;
795                 }

797                 return NULL;
798         }

800         if (get_rl_from_function(expr, &rl))
801                 goto found;

803         if (get_user_macro_rl(expr, &rl))
804                 goto found;

806         if (comes_from_skb_data(expr)) {
807                 rl = alloc_whole_rl(get_type(expr));
808                 goto found;
809         }

811         state = get_state_expr(my_id, expr);
812         if (state && estate_rl(state)) {
813                 rl = estate_rl(state);
814                 goto found;
815         }

817         if (expr->type == EXPR_CALL && db_returned_user_rl(expr, &rl))
818                 goto found;

820         if (is_array(expr)) {
821                 struct expression *array = get_array_base(expr);

823                 if (!get_state_expr(my_id, array)) {
824                         no_user_data_flag = 1;
825                         return NULL;
826                 }
827         }

829         if (expr->type == EXPR_PREOP && expr->op == '*' &&
830             is_user_rl(expr->unop)) {
831                 rl = var_to_absolute_rl(expr);
832                 goto found;
833         }

835         return NULL;
836 found:
837         user_data_flag = 1;
838         absolute_rl = var_to_absolute_rl(expr);
839         return clone_rl(rl_intersection(rl, absolute_rl));
840 }

842 int get_user_rl(struct expression *expr, struct range_list **rl)
843 {
844         user_data_flag = 0;
845         no_user_data_flag = 0;
846         custom_get_absolute_rl(expr, &var_user_rl, rl);
847         if (!user_data_flag || no_user_data_flag)
848                 *rl = NULL;

850         return !!*rl;
851 }
```



```
853 int get_user_rl_spammy(struct expression *expr, struct range_list **rl)
854 {
855         int ret;

857         option_spammy++;
858         ret = get_user_rl(expr, rl);
859         option_spammy--;

861         return ret;
862 }

864 int is_user_rl(struct expression *expr)
865 {
866         struct range_list *tmp;

868         return get_user_rl_spammy(expr, &tmp);
869 }

871 int get_user_rl_var_sym(const char *name, struct symbol *sym, struct range_list
872 {
873         struct smatch_state *state;

875         state = get_state(my_id, name, sym);
876         if (state && estate_rl(state)) {
877                 *rl = estate_rl(state);
878                 return 1;
879         }
880         return 0;
881 }

883 static void match_call_info(struct expression *expr)
884 {
885         struct range_list *rl;
886         struct expression *arg;
887         struct symbol *type;
888         int i = 0;

890         i = -1;
891         FOR_EACH_PTR(expr->args, arg) {
892                 i++;
893                 type = get_arg_type(expr->fn, i);

895                 if (!get_user_rl(arg, &rl))
896                         continue;

898                 rl = cast_rl(type, rl);
899                 sql_insert_caller_info(expr, USER_DATA3, i, "$", show_rl(rl));
900         } END_FOR_EACH_PTR(arg);
901 }

903 static int is_struct_ptr(struct symbol *sym)
904 {
905         struct symbol *type;

907         if (!sym)
908                 return 0;
909         type = get_real_base_type(sym);
910         if (!type || type->type != SYM_PTR)
911                 return 0;
912         type = get_real_base_type(type);
913         if (!type || type->type != SYM_STRUCT)
914                 return 0;
915         return 1;
916 }

918 static void struct_member_callback(struct expression *call, int param, char *pri
```

```
{
        struct smatch_state *state;
        struct range_list *rl;
        struct symbol *type;

        /*
         * Smatch uses a hack where if we get an unsigned long we say it's
         * both user data and it points to user data.  But if we pass it to a
         * function which takes an int, then it's just user data.  There's not
         * enough bytes for it to be a pointer.
         *
         */
        type = get_arg_type(call->fn, param);
        if (type && type_bits(type) < type_bits(&ptr_ctype))
                return;

        if (strcmp(sm->state->name, "") == 0)
                return;

        if (strcmp(printed_name, "*$") == 0 &&
            is_struct_ptr(sm->sym))
                return;

        state = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (!state || !estate_rl(state))
                rl = estate_rl(sm->state);
        else
                rl = rl_intersection(estate_rl(sm->state), estate_rl(state));

        sql_insert_caller_info(call, USER_DATA3, param, printed_name, show_rl(rl
}

static void set_param_user_data(const char *name, struct symbol *sym, char *key,
{
        struct range_list *rl = NULL;
        struct smatch_state *state;
        struct symbol *type;
        char fullname[256];

        if (strcmp(key, "*$") == 0)
                snprintf(fullname, sizeof(fullname), "*%s", name);
        else if (strncmp(key, "$", 1) == 0)
                snprintf(fullname, 256, "%s%s", name, key + 1);
        else
                return;

        type = get_member_type_from_key(symbol_expression(sym), key);

        /* if the caller passes a void pointer with user data */
        if (strcmp(key, "*$") == 0 && type && type != &void_ctype) {
                struct expression *expr = symbol_expression(sym);

                tag_as_user_data(expr);
                set_points_to_user_data(expr);
                return;
        }
        str_to_rl(type, value, &rl);
        state = alloc_estate_rl(rl);
        set_state(my_id, fullname, sym, state);
}

static void set_called(const char *name, struct symbol *sym, char *key, char *va
{
        set_state(my_call_id, "this_function", NULL, &called);
}
```



```
static void match_syscall_definition(struct symbol *sym)
{
        struct symbol *arg;
        char *macro;
        char *name;
        int is_syscall = 0;

        macro = get_macro_name(sym->pos);
        if (macro &&
            (strncmp("SYSCALL_DEFINE", macro, strlen("SYSCALL_DEFINE")) == 0 ||
             strncmp("COMPAT_SYSCALL_DEFINE", macro, strlen("COMPAT_SYSCALL_DEFI
                is_syscall = 1;

        name = get_function();
        if (!option_no_db && get_state(my_call_id, "this_function", NULL) != &ca
                if (name && strncmp(name, "sys_", 4) == 0)
                        is_syscall = 1;
        }

        if (name && strncmp(name, "compat_sys_", 11) == 0)
                is_syscall = 1;

        if (!is_syscall)
                return;

        FOR_EACH_PTR(sym->ctype.base_type->arguments, arg) {
                set_state(my_id, arg->ident->name, arg, alloc_estate_whole(get_r
        } END_FOR_EACH_PTR(arg);
}

static void set_to_user_data(struct expression *expr, char *key, char *value)
{
        char *name;
        struct symbol *sym;
        struct symbol *type;
        struct range_list *rl = NULL;

        type = get_member_type_from_key(expr, key);
        name = get_variable_from_key(expr, key, &sym);
        if (!name || !sym)
                goto free;

        call_results_to_rl(expr, type, value, &rl);

        set_state(my_id, name, sym, alloc_estate_rl(rl));
free:
        free_string(name);

}

static void returns_param_user_data(struct expression *expr, int param, char *ke
{
        struct expression *arg;
        struct expression *call;

        call = expr;
        while (call->type == EXPR_ASSIGNMENT)
                call = strip_expr(call->right);
        if (call->type != EXPR_CALL)
                return;

        if (!we_pass_user_data(call))
                return;

        if (param == -1) {
                if (expr->type != EXPR_ASSIGNMENT)
```

```
1051                        return;
1052                set_to_user_data(expr->left, key, value);
1053                return;
1054        }

1056        arg = get_argument_from_call_expr(call->args, param);
1057        if (!arg)
1058                return;
1059        set_to_user_data(arg, key, value);
1060 }

1062 static void returns_param_user_data_set(struct expression *expr, int param, char
1063 {
1064        struct expression *arg;

1066        func_gets_user_data = true;

1068        if (param == -1) {
1069                if (expr->type != EXPR_ASSIGNMENT)
1070                        return;
1071                if (strcmp(key, "*$") == 0) {
1072                        set_points_to_user_data(expr->left);
1073                        tag_as_user_data(expr->left);
1074                } else {
1075                        set_to_user_data(expr->left, key, value);
1076                }
1077                return;
1078        }

1080        while (expr->type == EXPR_ASSIGNMENT)
1081                expr = strip_expr(expr->right);
1082        if (expr->type != EXPR_CALL)
1083                return;

1085        arg = get_argument_from_call_expr(expr->args, param);
1086        if (!arg)
1087                return;
1088        set_to_user_data(arg, key, value);
1089 }

1091 static int has_empty_state(struct sm_state *sm)
1092 {
1093        struct sm_state *tmp;

1095        FOR_EACH_PTR(sm->possible, tmp) {
1096                if (!estate_rl(tmp->state))
1097                        return 1;
1098        } END_FOR_EACH_PTR(tmp);

1100        return 0;
1101 }

1103 static void param_set_to_user_data(int return_id, char *return_ranges, struct ex
1104 {
1105        struct sm_state *sm;
1106        struct smatch_state *start_state;
1107        struct range_list *rl;
1108        int param;
1109        char *return_str;
1110        const char *param_name;
1111        struct symbol *ret_sym;
1112        bool return_found = false;

1114        expr = strip_expr(expr);
1115        return_str = expr_to_str(expr);
1116        ret_sym = expr_to_sym(expr);
```



```
1118        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
1119                if (has_empty_state(sm))
1120                        continue;

1122                param = get_param_num_from_sym(sm->sym);
1123                if (param < 0)
1124                        continue;

1126                /* The logic here was that if we were passed in a user data then
1127                 * we don't record that.  It's like the difference between
1128                 * param_filter and param_set.  When I think about it, I'm not
1129                 * sure it actually works.  It's probably harmless because we
1130                 * checked earlier that we're not returning a parameter...
1131                 * Let's mark this as a TODO.
1132                 */
1133                start_state = get_state_stree(start_states, my_id, sm->name, sm-
1134                if (start_state && rl_equiv(estate_rl(sm->state), estate_rl(star
1135                        continue;

1137                param_name = get_param_name(sm);
1138                if (!param_name)
1139                        continue;
1140                if (strcmp(param_name, "$") == 0)  /* The -1 param is handled af
1141                        continue;

1143                sql_insert_return_states(return_id, return_ranges,
1144                                         func_gets_user_data ? USER_DATA3_SET :
1145                                         param, param_name, show_rl(estate_rl(sm
1146        } END_FOR_EACH_SM(sm);

1148        if (points_to_user_data(expr)) {
1149                sql_insert_return_states(return_id, return_ranges,
1150                                         (is_skb_data(expr) || !func_gets_user_d
1151                                         USER_DATA3_SET : USER_DATA3,
1152                                         -1, "*$", "");
1153                goto free_string;
1154        }


1157        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
1158                if (!ret_sym)
1159                        break;
1160                if (ret_sym != sm->sym)
1161                        continue;

1163                param_name = state_name_to_param_name(sm->name, return_str);
1164                if (!param_name)
1165                        continue;
1166                if (strcmp(param_name, "$") == 0)
1167                        return_found = true;
1168                sql_insert_return_states(return_id, return_ranges,
1169                                         func_gets_user_data ? USER_DATA3_SET :
1170                                         -1, param_name, show_rl(estate_rl(sm->s
1171        } END_FOR_EACH_SM(sm);


1174        if (!return_found && get_user_rl(expr, &rl)) {
1175                sql_insert_return_states(return_id, return_ranges,
1176                                         func_gets_user_data ? USER_DATA3_SET :
1177                                         -1, "$", show_rl(rl));
1178                goto free_string;
1179        }

1181 free_string:
1182        free_string(return_str);
```

```
1183 }

1185 static struct int_stack *gets_data_stack;
1186 static void match_function_def(struct symbol *sym)
1187 {
1188         func_gets_user_data = false;
1189 }

1191 static void match_inline_start(struct expression *expr)
1192 {
1193         push_int(&gets_data_stack, func_gets_user_data);
1194 }

1196 static void match_inline_end(struct expression *expr)
1197 {
1198         func_gets_user_data = pop_int(&gets_data_stack);
1199 }

1201 void register_kernel_user_data2(int id)
1202 {
1203         int i;

1205         my_id = id;

1207         if (option_project != PROJ_KERNEL)
1208                 return;

1210         add_hook(&match_function_def, FUNC_DEF_HOOK);
1211         add_hook(&match_inline_start, INLINE_FN_START);
1212         add_hook(&match_inline_end, INLINE_FN_END);

1214         add_hook(&save_start_states, AFTER_DEF_HOOK);
1215         add_hook(&free_start_states, AFTER_FUNC_HOOK);
1216         add_hook(&match_save_states, INLINE_FN_START);
1217         add_hook(&match_restore_states, INLINE_FN_END);

1219         add_unmatched_state_hook(my_id, &empty_state);
1220         add_extra_nomod_hook(&extra_nomod_hook);
1221         add_pre_merge_hook(my_id, &pre_merge_hook);
1222         add_merge_hook(my_id, &merge_estates);

1224         add_function_hook("copy_from_user", &match_user_copy, INT_PTR(0));
1225         add_function_hook("__copy_from_user", &match_user_copy, INT_PTR(0));
1226         add_function_hook("memcpy_fromiovec", &match_user_copy, INT_PTR(0));
1227         for (i = 0; i < ARRAY_SIZE(kstr_funcs); i++)
1228                 add_function_hook(kstr_funcs[i], &match_user_copy, INT_PTR(2));
1229         add_function_hook("usb_control_msg", &match_user_copy, INT_PTR(6));

1231         for (i = 0; i < ARRAY_SIZE(returns_user_data); i++) {
1232                 add_function_assign_hook(returns_user_data[i], &match_user_assig
1233                 add_function_hook(returns_user_data[i], &match_returns_user_rl,
1234         }

1236         add_function_hook("sscanf", &match_sscanf, NULL);

1238         add_hook(&match_syscall_definition, AFTER_DEF_HOOK);

1240         add_hook(&match_assign, ASSIGNMENT_HOOK);
1241         add_hook(&match_condition, CONDITION_HOOK);

1243         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
1244         add_member_info_callback(my_id, struct_member_callback);
1245         select_caller_info_hook(set_param_user_data, USER_DATA3);
1246         select_return_states_hook(USER_DATA3, &returns_param_user_data);
1247         select_return_states_hook(USER_DATA3_SET, &returns_param_user_data_set);
1248         add_split_return_callback(&param_set_to_user_data);
```



```
1249 }

1251 void register_kernel_user_data3(int id)
1252 {
1253         my_call_id = id;

1255         if (option_project != PROJ_KERNEL)
1256                 return;
1257         select_caller_info_hook(set_called, INTERNAL);
1258 }
```

```
**********************************************************
    2708 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_links.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Some helper functions for managing links.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static struct smatch_state *alloc_link(struct var_sym_list *links)
{
        struct smatch_state *state;
        static char buf[256];
        struct var_sym *tmp;
        int i;

        state = __alloc_smatch_state(0);

        i = 0;
        FOR_EACH_PTR(links, tmp) {
                if (!i++) {
                        snprintf(buf, sizeof(buf), "%s", tmp->var);
                } else {
                        append(buf, ", ", sizeof(buf));
                        append(buf, tmp->var, sizeof(buf));
                }
        } END_FOR_EACH_PTR(tmp);

        state->name = alloc_sname(buf);
        state->data = links;
        return state;
}

struct smatch_state *merge_link_states(struct smatch_state *s1, struct smatch_st
{
        struct var_sym_list *new_links;

        if (s1 == &undefined)
                return s2;
        if (s2 == &undefined)
                return s1;

        if (var_sym_lists_equiv(s1->data, s2->data))
                return s1;
```



```
        new_links = clone_var_sym_list(s1->data);
        merge_var_sym_list(&new_links, s2->data);

        return alloc_link(new_links);
}

void store_link(int link_id, const char *var, struct symbol *sym, const char *li
{

        struct smatch_state *old_state;
        struct var_sym_list *links;

        if (!cur_func_sym)
                return;

        old_state = get_state(link_id, var, sym);
        if (old_state)
                links = clone_var_sym_list(old_state->data);
        else
                links = NULL;

        add_var_sym(&links, link_name, link_sym);
        set_state(link_id, var, sym, alloc_link(links));
}

static void match_link_modify(struct sm_state *sm, struct expression *mod_expr)
{
        struct var_sym_list *links;
        struct var_sym *tmp;

        links = sm->state->data;

        FOR_EACH_PTR(links, tmp) {
                set_state(sm->owner - 1, tmp->var, tmp->sym, &undefined);
        } END_FOR_EACH_PTR(tmp);
        set_state(sm->owner, sm->name, sm->sym, &undefined);
}

void set_up_link_functions(int id, int link_id)
{
        if (id + 1 != link_id)
                sm_fatal("FATAL ERROR: links need to be registered directly afte

        add_merge_hook(link_id, &merge_link_states);
        add_modification_hook(link_id, &match_link_modify);
        // free link at the end of function
}
```

```
**********************************************************
    6044 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_local_values.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

/*
 * I'm going to store the states of local data at the end of each function.
 * Then at the end of the file, I'll combine the possible range lists for
 * each state and store the value in the on-disk database.
 *
 * One issue is that when I read the data back from the in-memory database at
 * the end of the file, then I don't have access to type information.  I'll just
 * cast everything to "long long" for now, I guess.  We'll see how that works.
 */

static char *db_vals;
static int get_vals(void *unused, int argc, char **argv, char **azColName)
{
        db_vals = alloc_string(argv[0]);
        return 0;
}

static int is_array_symbol(struct expression *expr)
{
        struct symbol *type;

        if (!expr || expr->type != EXPR_SYMBOL)
                return 0;
        type = get_type(expr);
        if (!type)
                return 0;
        if (type->type == SYM_ARRAY)
                return 1;
        return 0;
}

int get_local_rl(struct expression *expr, struct range_list **rl)
{
        char *name;
        struct range_list *tmp;
```



```
        if (!is_static(expr))
                return 0;
        if (is_array_symbol(expr))
                return 0;
        name = expr_to_var(expr);
        if (!name)
                return 0;

        db_vals = NULL;
        run_sql(get_vals, NULL,
                "select value from local_values where file = '%s' and variable =
                get_filename(), name);
        free_string(name);
        if (!db_vals)
                return 0;
        str_to_rl(&llong_ctype, db_vals, &tmp);
        *rl = cast_rl(get_type(expr), tmp);
        free_string(db_vals);

        return 1;
}

int get_local_max_helper(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;

        if (!get_local_rl(expr, &rl))
                return 0;
        *sval = rl_max(rl);
        return 1;
}

int get_local_min_helper(struct expression *expr, sval_t *sval)
{
        struct range_list *rl;

        if (!get_local_rl(expr, &rl))
                return 0;
        *sval = rl_min(rl);
        return 1;
}

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        return alloc_estate_empty();
}

static void extra_mod_hook(const char *name, struct symbol *sym, struct expressi
{
        struct smatch_state *old;
        struct smatch_state *new;

        if (!sym || !(sym->ctype.modifiers & MOD_STATIC))
                return;
        old = get_state(my_id, name, sym);
        if (old)
                new = merge_estates(old, state);
        else
                new = state;
        set_state(my_id, name, sym, new);
}

static void process_states(void)
{
        struct sm_state *sm;
        struct smatch_state *extra;
```

```
127         struct range_list *rl;

129         FOR_EACH_SM(__get_cur_stree(), sm) {
130                 if (sm->owner != my_id)
131                         continue;
132                 extra = get_state(SMATCH_EXTRA, sm->name, sm->sym);
133                 if (extra && estate_rl(extra))
134                         rl = rl_intersection(estate_rl(sm->state), estate_rl(ext
135                 else
136                         rl = estate_rl(sm->state);
137                 rl = cast_rl(&llong_ctype, rl);
138                 mem_sql(NULL, NULL,
139                         "insert into local_values values ('%s', '%s', '%s', %lu)
140                         get_filename(), sm->name, show_rl(rl),
141                         (unsigned long)sm->sym);
142         } END_FOR_EACH_SM(sm);
143 }

145 static int get_initial_value_sym(struct symbol *sym, char *name, sval_t *sval)
146 {
147         struct expression *expr_symbol, *deref, *tmp;
148         char *member_name;

150         if (!sym)
151                 return 0;

153         if (!sym->initializer) {
154                 *sval = sval_type_val(&llong_ctype, 0);
155                 return 1;
156         }
157         if (sym->initializer->type != EXPR_INITIALIZER)
158                 return get_value(sym->initializer, sval);

160         expr_symbol = symbol_expression(sym);
161         FOR_EACH_PTR(sym->initializer->expr_list, tmp) {
162                 if (tmp->type != EXPR_IDENTIFIER) /* how to handle arrays?? */
163                         continue;
164                 deref = member_expression(expr_symbol, '.', tmp->expr_ident);
165                 member_name = expr_to_var(deref);
166                 if (!member_name)
167                         continue;
168                 if (strcmp(name, member_name) == 0) {
169                         free_string(member_name);
170                         return get_value(tmp->ident_expression, sval);
171                 }
172                 free_string(member_name);
173         } END_FOR_EACH_PTR(tmp);

175         return 0;
176 }

178 static char *cur_name;
179 static struct symbol *cur_symbol;
180 static struct range_list *cur_rl;
181 static void add_current_local(void)
182 {
183         sval_t initial;

185         if (!get_initial_value_sym(cur_symbol, cur_name, &initial)) {
186                 free_string(cur_name);
187                 cur_name = NULL;
188                 cur_rl = NULL;
189                 return;
190         }
191         add_range(&cur_rl, initial, initial);
192         if (!is_whole_rl(cur_rl))
```



```
193                 sql_insert_local_values(cur_name, show_rl(cur_rl));
194         free_string(cur_name);
195         cur_name = NULL;
196         cur_rl = NULL;
197 }

199 static int save_final_values(void *unused, int argc, char **argv, char **azColNa
200 {
201         char *name = argv[0];
202         char *sym_str = argv[1];
203         char *value = argv[2];
204         struct range_list *rl;

206         if (!cur_name) {
207                 cur_name = alloc_string(name);
208                 cur_symbol = (struct symbol *)strtoul(sym_str, NULL, 10);
209         } else if (strcmp(cur_name, name) != 0) {
210                 add_current_local();
211                 cur_name = alloc_string(name);
212                 cur_symbol = (struct symbol *)strtoul(sym_str, NULL, 10);
213                 cur_rl = NULL;
214         }

216         str_to_rl(&llong_ctype, value, &rl);
217         cur_rl = rl_union(cur_rl, rl);

219         return 0;
220 }

222 static void match_end_file(struct symbol_list *sym_list)
223 {
224         mem_sql(save_final_values, NULL,
225                 "select distinct variable, symbol, value from local_values order
226         if (cur_name)
227                 add_current_local();
228 }

230 void register_local_values(int id)
231 {
232         my_id = id;

234         if (!option_info)
235                 return;

237         add_extra_mod_hook(&extra_mod_hook);
238         add_unmatched_state_hook(my_id, &unmatched_state);
239         add_merge_hook(my_id, &merge_estates);
240         all_return_states_hook(&process_states);
241         add_hook(match_end_file, END_FILE_HOOK);
242         mem_sql(NULL, NULL, "alter table local_values add column symbol integer;
243 }
```

```
**********************************************************
   38369 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_math.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "symbol.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static struct range_list *_get_rl(struct expression *expr, int implied, int *rec
static struct range_list *handle_variable(struct expression *expr, int implied,
static struct range_list *(*custom_handle_variable)(struct expression *expr);

static int get_implied_value_internal(struct expression *expr, sval_t *sval, int
static int get_absolute_rl_internal(struct expression *expr, struct range_list *

static sval_t zero  = {.type = &int_ctype, {.value = 0} };
static sval_t one   = {.type = &int_ctype, {.value = 1} };

struct range_list *rl_zero(void)
{
        static struct range_list *zero_perm;

        if (!zero_perm)
                zero_perm = clone_rl_permanent(alloc_rl(zero, zero));
        return zero_perm;
}

struct range_list *rl_one(void)
{
        static struct range_list *one_perm;

        if (!one_perm)
                one_perm = clone_rl_permanent(alloc_rl(one, one));

        return one_perm;
}

enum {
        RL_EXACT,
        RL_HARD,
        RL_FUZZY,
        RL_IMPLIED,
        RL_ABSOLUTE,
        RL_REAL_ABSOLUTE,
};
```



```
static struct range_list *last_stmt_rl(struct statement *stmt, int implied, int
{
        struct expression *expr;

        if (!stmt)
                return NULL;

        stmt = last_ptr_list((struct ptr_list *)stmt->stmts);
        if (stmt->type == STMT_LABEL) {
                if (stmt->label_statement &&
                    stmt->label_statement->type == STMT_EXPRESSION)
                        expr = stmt->label_statement->expression;
                else
                        return NULL;
        } else if (stmt->type == STMT_EXPRESSION) {
                expr = stmt->expression;
        } else {
                return NULL;
        }
        return _get_rl(expr, implied, recurse_cnt);
}

static struct range_list *handle_expression_statement_rl(struct expression *expr
{
        return last_stmt_rl(get_expression_statement(expr), implied, recurse_cnt
}

static struct range_list *handle_ampersand_rl(struct expression *expr, int impli
{
        struct range_list *rl;
        sval_t sval;

        if (implied == RL_EXACT || implied == RL_HARD)
                return NULL;
        if (get_mtag_sval(expr, &sval))
                return alloc_rl(sval, sval);
        if (get_address_rl(expr, &rl))
                return rl;
        return alloc_rl(valid_ptr_min_sval, valid_ptr_max_sval);
}

static struct range_list *handle_negate_rl(struct expression *expr, int implied,
{
        if (known_condition_true(expr->unop))
                return rl_zero();
        if (known_condition_false(expr->unop))
                return rl_one();

        if (implied == RL_EXACT)
                return NULL;

        if (implied_condition_true(expr->unop))
                return rl_zero();
        if (implied_condition_false(expr->unop))
                return rl_one();
        return alloc_rl(zero, one);
}

static struct range_list *handle_bitwise_negate(struct expression *expr, int imp
{
        struct range_list *rl;
        sval_t sval;

        rl = _get_rl(expr->unop, implied, recurse_cnt);
        if (!rl_to_sval(rl, &sval))
                return NULL;
```

```
127         sval = sval_preop(sval, '~');
128         sval_cast(get_type(expr->unop), sval);
129         return alloc_rl(sval, sval);
130 }

132 static struct range_list *handle_minus_preop(struct expression *expr, int implie
133 {
134         struct range_list *rl;
135         sval_t min, max;

137         rl = _get_rl(expr->unop, implied, recurse_cnt);
138         min = sval_preop(rl_max(rl), '-');
139         max = sval_preop(rl_min(rl), '-');
140         return alloc_rl(min, max);
141 }

143 static struct range_list *handle_preop_rl(struct expression *expr, int implied,
144 {
145         switch (expr->op) {
146         case '&':
147                 return handle_ampersand_rl(expr, implied, recurse_cnt);
148         case '!':
149                 return handle_negate_rl(expr, implied, recurse_cnt);
150         case '~':
151                 return handle_bitwise_negate(expr, implied, recurse_cnt);
152         case '-':
153                 return handle_minus_preop(expr, implied, recurse_cnt);
154         case '*':
155                 return handle_variable(expr, implied, recurse_cnt);
156         case '(':
157                 return handle_expression_statement_rl(expr, implied, recurse_cnt
158         default:
159                 return NULL;
160         }
161 }

163 static struct range_list *handle_divide_rl(struct expression *expr, int implied,
164 {
165         struct range_list *left_rl, *right_rl;
166         struct symbol *type;

168         type = get_type(expr);

170         left_rl = _get_rl(expr->left, implied, recurse_cnt);
171         left_rl = cast_rl(type, left_rl);
172         right_rl = _get_rl(expr->right, implied, recurse_cnt);
173         right_rl = cast_rl(type, right_rl);

175         if (!left_rl || !right_rl)
176                 return NULL;

178         if (implied != RL_REAL_ABSOLUTE) {
179                 if (is_whole_rl(left_rl) || is_whole_rl(right_rl))
180                         return NULL;
181         }

183         return rl_binop(left_rl, '/', right_rl);
184 }

186 static int handle_offset_subtraction(struct expression *expr)
187 {
188         struct expression *left, *right;
189         struct symbol *left_sym, *right_sym;
190         struct symbol *type;
191         int left_offset, right_offset;
```



```
193         type = get_type(expr);
194         if (!type || type->type != SYM_PTR)
195                 return -1;
196         type = get_real_base_type(type);
197         if (!type || (type_bits(type) != 8 && (type != &void_ctype)))
198                 return -1;

200         left = strip_expr(expr->left);
201         right = strip_expr(expr->right);

203         if (left->type != EXPR_PREOP || left->op != '&')
204                 return -1;
205         left = strip_expr(left->unop);

207         left_sym = expr_to_sym(left);
208         right_sym = expr_to_sym(right);
209         if (!left_sym || left_sym != right_sym)
210                 return -1;

212         left_offset = get_member_offset_from_deref(left);
213         if (right->type == EXPR_SYMBOL)
214                 right_offset = 0;
215         else {
216                 if (right->type != EXPR_PREOP || right->op != '&')
217                         return -1;
218                 right = strip_expr(right->unop);
219                 right_offset = get_member_offset_from_deref(right);
220         }
221         if (left_offset < 0 || right_offset < 0)
222                 return -1;

224         return left_offset - right_offset;
225 }

227 static struct range_list *handle_subtract_rl(struct expression *expr, int implie
228 {
229         struct symbol *type;
230         struct range_list *left_orig, *right_orig;
231         struct range_list *left_rl, *right_rl;
232         sval_t max, min, tmp;
233         int comparison;
234         int offset;

236         type = get_type(expr);

238         offset = handle_offset_subtraction(expr);
239         if (offset >= 0) {
240                 tmp.type = type;
241                 tmp.value = offset;

243                 return alloc_rl(tmp, tmp);
244         }

246         comparison = get_comparison(expr->left, expr->right);

248         left_orig = _get_rl(expr->left, implied, recurse_cnt);
249         left_rl = cast_rl(type, left_orig);
250         right_orig = _get_rl(expr->right, implied, recurse_cnt);
251         right_rl = cast_rl(type, right_orig);

253         if ((!left_rl || !right_rl) &&
254             (implied == RL_EXACT || implied == RL_HARD || implied == RL_FUZZY))
255                 return NULL;

257         if (!left_rl)
258                 left_rl = alloc_whole_rl(type);
```

```
259         if (!right_rl)
260                 right_rl = alloc_whole_rl(type);
261
262         /* negative values complicate everything fix this later */
263         if (sval_is_negative(rl_min(right_rl)))
264                 return NULL;
265         max = rl_max(left_rl);
266         min = sval_type_min(type);
267
268         switch (comparison) {
269         case '>':
270         case SPECIAL_UNSIGNED_GT:
271                 min = sval_type_val(type, 1);
272                 max = rl_max(left_rl);
273                 break;
274         case SPECIAL_GTE:
275         case SPECIAL_UNSIGNED_GTE:
276                 min = sval_type_val(type, 0);
277                 max = rl_max(left_rl);
278                 break;
279         case SPECIAL_EQUAL:
280                 min = sval_type_val(type, 0);
281                 max = sval_type_val(type, 0);
282                 break;
283         case '<':
284         case SPECIAL_UNSIGNED_LT:
285                 max = sval_type_val(type, -1);
286                 break;
287         case SPECIAL_LTE:
288         case SPECIAL_UNSIGNED_LTE:
289                 max = sval_type_val(type, 0);
290                 break;
291         default:
292                 if (!left_orig || !right_orig)
293                         return NULL;
294                 return rl_binop(left_rl, '-', right_rl);
295         }
296
297         if (!sval_binop_overflows(rl_min(left_rl), '-', rl_max(right_rl))) {
298                 tmp = sval_binop(rl_min(left_rl), '-', rl_max(right_rl));
299                 if (sval_cmp(tmp, min) > 0)
300                         min = tmp;
301         }
302
303         if (!sval_is_max(rl_max(left_rl))) {
304                 tmp = sval_binop(rl_max(left_rl), '-', rl_min(right_rl));
305                 if (sval_cmp(tmp, max) < 0)
306                         max = tmp;
307         }
308
309         if (sval_is_min(min) && sval_is_max(max))
310                 return NULL;
311
312         return cast_rl(type, alloc_rl(min, max));
313 }
314
315 static struct range_list *handle_mod_rl(struct expression *expr, int implied, in
316 {
317         struct range_list *rl;
318         sval_t left, right, sval;
319
320         if (implied == RL_EXACT) {
321                 if (!get_implied_value(expr->right, &right))
322                         return NULL;
323                 if (!get_implied_value(expr->left, &left))
324                         return NULL;
```



```
325                 sval = sval_binop(left, '%', right);
326                 return alloc_rl(sval, sval);
327         }
328         /* if we can't figure out the right side it's probably hopeless */
329         if (!get_implied_value_internal(expr->right, &right, recurse_cnt))
330                 return NULL;
331
332         right = sval_cast(get_type(expr), right);
333         right.value--;
334
335         rl = _get_rl(expr->left, implied, recurse_cnt);
336         if (rl && rl_max(rl).uvalue < right.uvalue)
337                 right.uvalue = rl_max(rl).uvalue;
338
339         return alloc_rl(sval_cast(right.type, zero), right);
340 }
341
342 static sval_t sval_lowest_set_bit(sval_t sval)
343 {
344         int i;
345         int found = 0;
346
347         for (i = 0; i < 64; i++) {
348                 if (sval.uvalue & 1ULL << i) {
349                         if (!found++)
350                                 continue;
351                         sval.uvalue &= ~(1ULL << i);
352                 }
353         }
354         return sval;
355 }
356
357 static struct range_list *handle_bitwise_AND(struct expression *expr, int implie
358 {
359         struct symbol *type;
360         struct range_list *left_rl, *right_rl;
361         sval_t known;
362         int new_recurse;
363
364         if (implied != RL_IMPLIED && implied != RL_ABSOLUTE && implied != RL_REA
365                 return NULL;
366
367         type = get_type(expr);
368
369         if (get_implied_value_internal(expr->left, &known, recurse_cnt)) {
370                 sval_t min;
371
372                 min = sval_lowest_set_bit(known);
373                 left_rl = alloc_rl(min, known);
374                 left_rl = cast_rl(type, left_rl);
375                 add_range(&left_rl, sval_type_val(type, 0), sval_type_val(type,
376         } else {
377                 left_rl = _get_rl(expr->left, implied, recurse_cnt);
378                 if (left_rl) {
379                         left_rl = cast_rl(type, left_rl);
380                         left_rl = alloc_rl(sval_type_val(type, 0), rl_max(left_r
381                 } else {
382                         if (implied == RL_HARD)
383                                 return NULL;
384                         left_rl = alloc_whole_rl(type);
385                 }
386         }
387
388         new_recurse = *recurse_cnt;
389         if (*recurse_cnt >= 200)
390                 new_recurse = 100;  /* Let's try super hard to get the mask */
```

```
391         if (get_implied_value_internal(expr->right, &known, &new_recurse)) {
392                 sval_t min, left_max, mod;

394                 *recurse_cnt = new_recurse;

396                 min = sval_lowest_set_bit(known);
397                 right_rl = alloc_rl(min, known);
398                 right_rl = cast_rl(type, right_rl);
399                 add_range(&right_rl, sval_type_val(type, 0), sval_type_val(type,

401                 if (min.value != 0) {
402                         left_max = rl_max(left_rl);
403                         mod = sval_binop(left_max, '%', min);
404                         if (mod.value) {
405                                 left_max = sval_binop(left_max, '-', mod);
406                                 left_max.value++;
407                                 if (left_max.value > 0 && sval_cmp(left_max, rl_
408                                         left_rl = remove_range(left_rl, left_max
409                         }
410                 }
411         } else {
412                 right_rl = _get_rl(expr->right, implied, recurse_cnt);
413                 if (right_rl) {
414                         right_rl = cast_rl(type, right_rl);
415                         right_rl = alloc_rl(sval_type_val(type, 0), rl_max(right
416                 } else {
417                         if (implied == RL_HARD)
418                                 return NULL;
419                         right_rl = alloc_whole_rl(type);
420                 }
421         }

423         return rl_intersection(left_rl, right_rl);
424 }

426 static struct range_list *use_rl_binop(struct expression *expr, int implied, int
427 {
428         struct symbol *type;
429         struct range_list *left_rl, *right_rl;

431         if (implied != RL_IMPLIED && implied != RL_ABSOLUTE && implied != RL_REA
432                 return NULL;

434         type = get_type(expr);

436         get_absolute_rl_internal(expr->left, &left_rl, recurse_cnt);
437         get_absolute_rl_internal(expr->right, &right_rl, recurse_cnt);
438         left_rl = cast_rl(type, left_rl);
439         right_rl = cast_rl(type, right_rl);
440         if (!left_rl || !right_rl)
441                 return NULL;

443         return rl_binop(left_rl, expr->op, right_rl);
444 }

446 static struct range_list *handle_right_shift(struct expression *expr, int implie
447 {
448         struct range_list *left_rl;
449         sval_t right;
450         sval_t min, max;

452         if (implied == RL_EXACT || implied == RL_HARD)
453                 return NULL;

455         left_rl = _get_rl(expr->left, implied, recurse_cnt);
456         if (left_rl) {
```



```
457                 max = rl_max(left_rl);
458                 min = rl_min(left_rl);
459         } else {
460                 if (implied == RL_FUZZY)
461                         return NULL;
462                 max = sval_type_max(get_type(expr->left));
463                 min = sval_type_val(get_type(expr->left), 0);
464         }

466         if (get_implied_value_internal(expr->right, &right, recurse_cnt)) {
467                 min = sval_binop(min, SPECIAL_RIGHTSHIFT, right);
468                 max = sval_binop(max, SPECIAL_RIGHTSHIFT, right);
469         } else if (!sval_is_negative(min)) {
470                 min.value = 0;
471                 max = sval_type_max(max.type);
472         } else {
473                 return NULL;
474         }

476         return alloc_rl(min, max);
477 }

479 static struct range_list *handle_left_shift(struct expression *expr, int implied
480 {
481         struct range_list *left_rl, *res;
482         sval_t right;
483         sval_t min, max;
484         int add_zero = 0;

486         if (implied == RL_EXACT || implied == RL_HARD)
487                 return NULL;
488         /* this is hopeless without the right side */
489         if (!get_implied_value_internal(expr->right, &right, recurse_cnt))
490                 return NULL;
491         left_rl = _get_rl(expr->left, implied, recurse_cnt);
492         if (left_rl) {
493                 max = rl_max(left_rl);
494                 min = rl_min(left_rl);
495                 if (min.value == 0) {
496                         min.value = 1;
497                         add_zero = 1;
498                 }
499         } else {
500                 if (implied == RL_FUZZY)
501                         return NULL;
502                 max = sval_type_max(get_type(expr->left));
503                 min = sval_type_val(get_type(expr->left), 1);
504                 add_zero = 1;
505         }

507         max = sval_binop(max, SPECIAL_LEFTSHIFT, right);
508         min = sval_binop(min, SPECIAL_LEFTSHIFT, right);
509         res = alloc_rl(min, max);
510         if (add_zero)
511                 res = rl_union(res, rl_zero());
512         return res;
513 }

515 static struct range_list *handle_known_binop(struct expression *expr)
516 {
517         sval_t left, right;

519         if (!get_value(expr->left, &left))
520                 return NULL;
521         if (!get_value(expr->right, &right))
522                 return NULL;
```

```
523         left = sval_binop(left, expr->op, right);
524         return alloc_rl(left, left);
525 }

527 static int has_actual_ranges(struct range_list *rl)
528 {
529         struct data_range *tmp;

531         FOR_EACH_PTR(rl, tmp) {
532                 if (sval_cmp(tmp->min, tmp->max) != 0)
533                         return 1;
534         } END_FOR_EACH_PTR(tmp);
535         return 0;
536 }

538 static struct range_list *handle_implied_binop(struct range_list *left_rl, int o
539 {
540         struct range_list *res_rl;
541         struct data_range *left_drange, *right_drange;
542         sval_t res;

544         if (!left_rl || !right_rl)
545                 return NULL;
546         if (has_actual_ranges(left_rl))
547                 return NULL;
548         if (has_actual_ranges(right_rl))
549                 return NULL;

551         if (ptr_list_size((struct ptr_list *)left_rl) * ptr_list_size((struct pt
552                 return NULL;

554         res_rl = NULL;

556         FOR_EACH_PTR(left_rl, left_drange) {
557                 FOR_EACH_PTR(right_rl, right_drange) {
558                         if ((op == '%' || op == '/') &&
559                             right_drange->min.value == 0)
560                                 return NULL;
561                         res = sval_binop(left_drange->min, op, right_drange->min
562                         add_range(&res_rl, res, res);
563                 } END_FOR_EACH_PTR(right_drange);
564         } END_FOR_EACH_PTR(left_drange);

566         return res_rl;
567 }

569 static struct range_list *handle_binop_rl(struct expression *expr, int implied,
570 {
571         struct smatch_state *state;
572         struct symbol *type;
573         struct range_list *left_rl, *right_rl, *rl;
574         sval_t min, max;

576         rl = handle_known_binop(expr);
577         if (rl)
578                 return rl;
579         if (implied == RL_EXACT)
580                 return NULL;

582         if (custom_handle_variable) {
583                 rl = custom_handle_variable(expr);
584                 if (rl)
585                         return rl;
586         }

588         state = get_extra_state(expr);
```



```
589         if (state && !is_whole_rl(estate_rl(state))) {
590                 if (implied != RL_HARD || estate_has_hard_max(state))
591                         return clone_rl(estate_rl(state));
592         }

594         type = get_type(expr);
595         left_rl = _get_rl(expr->left, implied, recurse_cnt);
596         left_rl = cast_rl(type, left_rl);
597         right_rl = _get_rl(expr->right, implied, recurse_cnt);
598         right_rl = cast_rl(type, right_rl);

600         if (!left_rl && !right_rl)
601                 return NULL;

603         rl = handle_implied_binop(left_rl, expr->op, right_rl);
604         if (rl)
605                 return rl;

607         switch (expr->op) {
608         case '%':
609                 return handle_mod_rl(expr, implied, recurse_cnt);
610         case '&':
611                 return handle_bitwise_AND(expr, implied, recurse_cnt);
612         case '|':
613         case '^':
614                 return use_rl_binop(expr, implied, recurse_cnt);
615         case SPECIAL_RIGHTSHIFT:
616                 return handle_right_shift(expr, implied, recurse_cnt);
617         case SPECIAL_LEFTSHIFT:
618                 return handle_left_shift(expr, implied, recurse_cnt);
619         case '-':
620                 return handle_subtract_rl(expr, implied, recurse_cnt);
621         case '/':
622                 return handle_divide_rl(expr, implied, recurse_cnt);
623         }

625         if (!left_rl || !right_rl)
626                 return NULL;

628         if (sval_binop_overflows(rl_min(left_rl), expr->op, rl_min(right_rl)))
629                 return NULL;
630         if (sval_binop_overflows(rl_max(left_rl), expr->op, rl_max(right_rl)))
631                 return NULL;

633         min = sval_binop(rl_min(left_rl), expr->op, rl_min(right_rl));
634         max = sval_binop(rl_max(left_rl), expr->op, rl_max(right_rl));

636         return alloc_rl(min, max);
637 }

639 static int do_comparison(struct expression *expr)
640 {
641         struct range_list *left_ranges = NULL;
642         struct range_list *right_ranges = NULL;
643         int poss_true, poss_false;
644         struct symbol *type;

646         type = get_type(expr);
647         get_absolute_rl(expr->left, &left_ranges);
648         get_absolute_rl(expr->right, &right_ranges);

650         left_ranges = cast_rl(type, left_ranges);
651         right_ranges = cast_rl(type, right_ranges);

653         poss_true = possibly_true_rl(left_ranges, expr->op, right_ranges);
654         poss_false = possibly_false_rl(left_ranges, expr->op, right_ranges);
```

```
656         if (!poss_true && !poss_false)
657                 return 0x0;
658         if (poss_true && !poss_false)
659                 return 0x1;
660         if (!poss_true && poss_false)
661                 return 0x2;
662         return 0x3;
663 }

665 static struct range_list *handle_comparison_rl(struct expression *expr, int impl
666 {
667         sval_t left, right;
668         int res;

670         if (expr->op == SPECIAL_EQUAL && expr->left->type == EXPR_TYPE) {
671                 struct symbol *left, *right;

673                 left = get_real_base_type(expr->left->symbol);
674                 right = get_real_base_type(expr->left->symbol);
675                 if (left == right)
676                         return rl_one();
677                 return rl_zero();
678         }

680         if (get_value(expr->left, &left) && get_value(expr->right, &right)) {
681                 struct data_range tmp_left, tmp_right;

683                 tmp_left.min = left;
684                 tmp_left.max = left;
685                 tmp_right.min = right;
686                 tmp_right.max = right;
687                 if (true_comparison_range(&tmp_left, expr->op, &tmp_right))
688                         return rl_one();
689                 return rl_zero();
690         }

692         if (implied == RL_EXACT)
693                 return NULL;

695         res = do_comparison(expr);
696         if (res == 1)
697                 return rl_one();
698         if (res == 2)
699                 return rl_zero();

701         return alloc_rl(zero, one);
702 }

704 static struct range_list *handle_logical_rl(struct expression *expr, int implied
705 {
706         sval_t left, right;
707         int left_known = 0;
708         int right_known = 0;

710         if (implied == RL_EXACT) {
711                 if (get_value(expr->left, &left))
712                         left_known = 1;
713                 if (get_value(expr->right, &right))
714                         right_known = 1;
715         } else {
716                 if (get_implied_value_internal(expr->left, &left, recurse_cnt))
717                         left_known = 1;
718                 if (get_implied_value_internal(expr->right, &right, recurse_cnt)
719                         right_known = 1;
720         }
```



```
722         switch (expr->op) {
723         case SPECIAL_LOGICAL_OR:
724                 if (left_known && left.value)
725                         return rl_one();
726                 if (right_known && right.value)
727                         return rl_one();
728                 if (left_known && right_known)
729                         return rl_zero();
730                 break;
731         case SPECIAL_LOGICAL_AND:
732                 if (left_known && right_known) {
733                         if (left.value && right.value)
734                                 return rl_one();
735                         return rl_zero();
736                 }
737                 break;
738         default:
739                 return NULL;
740         }

742         if (implied == RL_EXACT)
743                 return NULL;

745         return alloc_rl(zero, one);
746 }

748 static struct range_list *handle_conditional_rl(struct expression *expr, int imp
749 {
750         struct expression *cond_true;
751         struct range_list *true_rl, *false_rl;
752         struct symbol *type;
753         int final_pass_orig = final_pass;

755         cond_true = expr->cond_true;
756         if (!cond_true)
757                 cond_true = expr->conditional;

759         if (known_condition_true(expr->conditional))
760                 return _get_rl(cond_true, implied, recurse_cnt);
761         if (known_condition_false(expr->conditional))
762                 return _get_rl(expr->cond_false, implied, recurse_cnt);

764         if (implied == RL_EXACT)
765                 return NULL;

767         if (implied_condition_true(expr->conditional))
768                 return _get_rl(cond_true, implied, recurse_cnt);
769         if (implied_condition_false(expr->conditional))
770                 return _get_rl(expr->cond_false, implied, recurse_cnt);


773         /* this becomes a problem with deeply nested conditional statements */
774         if (low_on_memory())
775                 return NULL;

777         type = get_type(expr);

779         __push_fake_cur_stree();
780         final_pass = 0;
781         __split_whole_condition(expr->conditional);
782         true_rl = _get_rl(cond_true, implied, recurse_cnt);
783         __push_true_states();
784         __use_false_states();
785         false_rl = _get_rl(expr->cond_false, implied, recurse_cnt);
786         __merge_true_states();
```

```
787         __free_fake_cur_stree();
788         final_pass = final_pass_orig;

790         if (!true_rl || !false_rl)
791                 return NULL;
792         true_rl = cast_rl(type, true_rl);
793         false_rl = cast_rl(type, false_rl);

795         return rl_union(true_rl, false_rl);
796 }

798 static int get_fuzzy_max_helper(struct expression *expr, sval_t *max)
799 {
800         struct smatch_state *state;
801         sval_t sval;

803         if (get_hard_max(expr, &sval)) {
804                 *max = sval;
805                 return 1;
806         }

808         state = get_extra_state(expr);
809         if (!state || !estate_has_fuzzy_max(state))
810                 return 0;
811         *max = sval_cast(get_type(expr), estate_get_fuzzy_max(state));
812         return 1;
813 }

815 static int get_fuzzy_min_helper(struct expression *expr, sval_t *min)
816 {
817         struct smatch_state *state;
818         sval_t sval;

820         state = get_extra_state(expr);
821         if (!state || !estate_rl(state))
822                 return 0;

824         sval = estate_min(state);
825         if (sval_is_negative(sval) && sval_is_min(sval))
826                 return 0;

828         if (sval_is_max(sval))
829                 return 0;

831         *min = sval_cast(get_type(expr), sval);
832         return 1;
833 }

835 int get_const_value(struct expression *expr, sval_t *sval)
836 {
837         struct symbol *sym;
838         sval_t right;

840         if (expr->type != EXPR_SYMBOL || !expr->symbol)
841                 return 0;
842         sym = expr->symbol;
843         if (!(sym->ctype.modifiers & MOD_CONST))
844                 return 0;
845         if (get_value(sym->initializer, &right)) {
846                 *sval = sval_cast(get_type(expr), right);
847                 return 1;
848         }
849         return 0;
850 }

852 struct range_list *var_to_absolute_rl(struct expression *expr)
```



```
853 {
854         struct smatch_state *state;
855         struct range_list *rl;

857         state = get_extra_state(expr);
858         if (!state || is_whole_rl(estate_rl(state))) {
859                 state = get_real_absolute_state(expr);
860                 if (state && state->data && !estate_is_whole(state))
861                         return clone_rl(estate_rl(state));
862                 if (get_local_rl(expr, &rl) && !is_whole_rl(rl))
863                         return rl;
864                 if (get_mtag_rl(expr, &rl))
865                         return rl;
866                 if (get_db_type_rl(expr, &rl) && !is_whole_rl(rl))
867                         return rl;
868                 return alloc_whole_rl(get_type(expr));
869         }
870         /* err on the side of saying things are possible */
871         if (!estate_rl(state))
872                 return alloc_whole_rl(get_type(expr));
873         return clone_rl(estate_rl(state));
874 }

876 static struct range_list *handle_variable(struct expression *expr, int implied,
877 {
878         struct smatch_state *state;
879         struct range_list *rl;
880         sval_t sval, min, max;
881         struct symbol *type;

883         if (get_const_value(expr, &sval))
884                 return alloc_rl(sval, sval);

886         if (custom_handle_variable) {
887                 rl = custom_handle_variable(expr);
888                 if (!rl)
889                         return var_to_absolute_rl(expr);
890                 return rl;
891         }

893         if (implied == RL_EXACT)
894                 return NULL;

896         if (get_mtag_sval(expr, &sval))
897                 return alloc_rl(sval, sval);

899         type = get_type(expr);
900         if (type && type->type == SYM_FN)
901                 return alloc_rl(fn_ptr_min, fn_ptr_max);

903         switch (implied) {
904         case RL_HARD:
905         case RL_IMPLIED:
906         case RL_ABSOLUTE:
907                 state = get_extra_state(expr);
908                 if (!state || !state->data) {
909                         if (implied == RL_HARD)
910                                 return NULL;
911                         if (get_local_rl(expr, &rl))
912                                 return rl;
913                         if (get_mtag_rl(expr, &rl))
914                                 return rl;
915                         if (get_db_type_rl(expr, &rl))
916                                 return rl;
917                         if (is_array(expr) && get_array_rl(expr, &rl))
918                                 return rl;
```

```
919                     return NULL;
920             }
921             if (implied == RL_HARD && !estate_has_hard_max(state))
922                     return NULL;
923             return clone_rl(estate_rl(state));
924     case RL_REAL_ABSOLUTE: {
925             struct smatch_state *abs_state;
926
927             state = get_extra_state(expr);
928             abs_state = get_real_absolute_state(expr);
929
930             if (estate_rl(state) && estate_rl(abs_state)) {
931                     return clone_rl(rl_intersection(estate_rl(state),
932                                                     estate_rl(abs_state)));
933             } else if (estate_rl(state)) {
934                     return clone_rl(estate_rl(state));
935             } else if (estate_is_empty(state)) {
936                     /*
937                      * FIXME: we don't handle empty extra states correctly.
938                      *
939                      * The real abs rl is supposed to be filtered by the
940                      * extra state if there is one.  We don't bother keeping
941                      * the abs state in sync all the time because we know it
942                      * will be filtered later.
943                      *
944                      * It's not totally obvious to me how they should be
945                      * handled.  Perhaps we should take the whole rl and
946                      * filter by the imaginary states.  Perhaps we should
947                      * just go with the empty state.
948                      *
949                      * Anyway what we currently do is return NULL here and
950                      * that gets translated into the whole range in
951                      * get_real_absolute_rl().
952                      *
953                      */
954                     return NULL;
955             } else if (estate_rl(abs_state)) {
956                     return clone_rl(estate_rl(abs_state));
957             }
958
959             if (get_local_rl(expr, &rl))
960                     return rl;
961             if (get_mtag_rl(expr, &rl))
962                     return rl;
963             if (get_db_type_rl(expr, &rl))
964                     return rl;
965             if (is_array(expr) && get_array_rl(expr, &rl))
966                     return rl;
967             return NULL;
968     }
969     case RL_FUZZY:
970             if (!get_fuzzy_min_helper(expr, &min))
971                     min = sval_type_min(get_type(expr));
972             if (!get_fuzzy_max_helper(expr, &max))
973                     return NULL;
974             /* fuzzy ranges are often inverted */
975             if (sval_cmp(min, max) > 0) {
976                     sval = min;
977                     min = max;
978                     max = sval;
979             }
980             return alloc_rl(min, max);
981     }
982     return NULL;
983 }
```



```
985 static sval_t handle_sizeof(struct expression *expr)
986 {
987         struct symbol *sym;
988         sval_t ret;
989
990         ret = sval_blank(expr);
991         sym = expr->cast_type;
992         if (!sym) {
993                 sym = evaluate_expression(expr->cast_expression);
994                 if (!sym) {
995                         __silence_warnings_for_stmt = true;
996                         sym = &int_ctype;
997                 }
998 #if 0
999                 /*
1000                  * Expressions of restricted types will possibly get
1001                  * promoted - check that here.  I'm not sure how this works,
1002                  * the problem is that sizeof(le16) shouldn't be promoted and
1003                  * the original code did that...  Let's if zero this out and
1004                  * see what breaks.
1005                  */
1006
1007                 if (is_restricted_type(sym)) {
1008                         if (type_bits(sym) < bits_in_int)
1009                                 sym = &int_ctype;
1010                 }
1011 #endif
1012                 if (is_fouled_type(sym))
1013                         sym = &int_ctype;
1014         }
1015         examine_symbol_type(sym);
1016
1017         ret.type = size_t_ctype;
1018         if (type_bits(sym) <= 0) /* sizeof(void) */ {
1019                 if (get_real_base_type(sym) == &void_ctype)
1020                         ret.value = 1;
1021                 else
1022                         ret.value = 0;
1023         } else
1024                 ret.value = type_bytes(sym);
1025
1026         return ret;
1027 }
1028
1029 static struct range_list *handle_strlen(struct expression *expr, int implied, in
1030 {
1031         struct range_list *rl;
1032         struct expression *arg, *tmp;
1033         sval_t tag;
1034         sval_t ret = { .type = &ulong_ctype };
1035
1036         if (implied == RL_EXACT)
1037                 return NULL;
1038
1039         arg = get_argument_from_call_expr(expr->args, 0);
1040         if (!arg)
1041                 return NULL;
1042         if (arg->type == EXPR_STRING) {
1043                 ret.value = arg->string->length - 1;
1044                 return alloc_rl(ret, ret);
1045         }
1046         if (get_implied_value(arg, &tag) &&
1047             (tmp = fake_string_from_mtag(tag.uvalue))) {
1048                 ret.value = tmp->string->length - 1;
1049                 return alloc_rl(ret, ret);
1050         }
```

```
1052         if (implied == RL_HARD || implied == RL_FUZZY)
1053                 return NULL;
1054
1055         if (get_implied_return(expr, &rl))
1056                 return rl;
1057
1058         return NULL;
1059 }
1060
1061 static struct range_list *handle_builtin_constant_p(struct expression *expr, int
1062 {
1063         struct expression *arg;
1064         struct range_list *rl;
1065         sval_t sval;
1066
1067         arg = get_argument_from_call_expr(expr->args, 0);
1068         rl = _get_rl(arg, RL_EXACT, recurse_cnt);
1069         if (rl_to_sval(rl, &sval))
1070                 return rl_one();
1071         return rl_zero();
1072 }
1073
1074 static struct range_list *handle__builtin_choose_expr(struct expression *expr, i
1075 {
1076         struct expression *const_expr, *expr1, *expr2;
1077         sval_t sval;
1078
1079         const_expr = get_argument_from_call_expr(expr->args, 0);
1080         expr1 = get_argument_from_call_expr(expr->args, 1);
1081         expr2 = get_argument_from_call_expr(expr->args, 2);
1082
1083         if (!get_value(const_expr, &sval) || !expr1 || !expr2)
1084                 return NULL;
1085         if (sval.value)
1086                 return _get_rl(expr1, implied, recurse_cnt);
1087         return _get_rl(expr2, implied, recurse_cnt);
1088 }
1089
1090 static struct range_list *handle_call_rl(struct expression *expr, int implied, i
1091 {
1092         struct range_list *rl;
1093
1094         if (sym_name_is("__builtin_constant_p", expr->fn))
1095                 return handle_builtin_constant_p(expr, implied, recurse_cnt);
1096
1097         if (sym_name_is("__builtin_choose_expr", expr->fn))
1098                 return handle__builtin_choose_expr(expr, implied, recurse_cnt);
1099
1100         if (sym_name_is("__builtin_expect", expr->fn) ||
1101             sym_name_is("__builtin_bswap16", expr->fn) ||
1102             sym_name_is("__builtin_bswap32", expr->fn) ||
1103             sym_name_is("__builtin_bswap64", expr->fn)) {
1104                 struct expression *arg;
1105
1106                 arg = get_argument_from_call_expr(expr->args, 0);
1107                 return _get_rl(arg, implied, recurse_cnt);
1108         }
1109
1110         if (sym_name_is("strlen", expr->fn))
1111                 return handle_strlen(expr, implied, recurse_cnt);
1112
1113         if (implied == RL_EXACT || implied == RL_HARD || implied == RL_FUZZY)
1114                 return NULL;
1115
1116         if (custom_handle_variable) {
```



```
1117                 rl = custom_handle_variable(expr);
1118                 if (rl)
1119                         return rl;
1120         }
1121
1122         if (get_implied_return(expr, &rl))
1123                 return rl;
1124         return db_return_vals(expr);
1125 }
1126
1127 static struct range_list *handle_cast(struct expression *expr, int implied, int
1128 {
1129         struct range_list *rl;
1130         struct symbol *type;
1131
1132         type = get_type(expr);
1133         rl = _get_rl(expr->cast_expression, implied, recurse_cnt);
1134         if (rl)
1135                 return cast_rl(type, rl);
1136         if (implied == RL_ABSOLUTE || implied == RL_REAL_ABSOLUTE)
1137                 return alloc_whole_rl(type);
1138         if (implied == RL_IMPLIED && type &&
1139             type_bits(type) > 0 && type_bits(type) < 32)
1140                 return alloc_whole_rl(type);
1141         return NULL;
1142 }
1143
1144 static struct range_list *_get_rl(struct expression *expr, int implied, int *rec
1145 {
1146         struct range_list *rl;
1147         struct symbol *type;
1148         sval_t sval;
1149
1150         type = get_type(expr);
1151         expr = strip_parens(expr);
1152         if (!expr)
1153                 return NULL;
1154
1155         if (++(*recurse_cnt) >= 200)
1156                 return NULL;
1157
1158         switch(expr->type) {
1159         case EXPR_CAST:
1160         case EXPR_FORCE_CAST:
1161         case EXPR_IMPLIED_CAST:
1162                 rl = handle_cast(expr, implied, recurse_cnt);
1163                 goto out_cast;
1164         }
1165
1166         expr = strip_expr(expr);
1167         if (!expr)
1168                 return NULL;
1169
1170         switch (expr->type) {
1171         case EXPR_VALUE:
1172                 sval = sval_from_val(expr, expr->value);
1173                 rl = alloc_rl(sval, sval);
1174                 break;
1175         case EXPR_PREOP:
1176                 rl = handle_preop_rl(expr, implied, recurse_cnt);
1177                 break;
1178         case EXPR_POSTOP:
1179                 rl = _get_rl(expr->unop, implied, recurse_cnt);
1180                 break;
1181         case EXPR_BINOP:
1182                 rl = handle_binop_rl(expr, implied, recurse_cnt);
```

```
1183                    break;
1184            case EXPR_COMPARE:
1185                    rl = handle_comparison_rl(expr, implied, recurse_cnt);
1186                    break;
1187            case EXPR_LOGICAL:
1188                    rl = handle_logical_rl(expr, implied, recurse_cnt);
1189                    break;
1190            case EXPR_PTRSIZEOF:
1191            case EXPR_SIZEOF:
1192                    sval = handle_sizeof(expr);
1193                    rl = alloc_rl(sval, sval);
1194                    break;
1195            case EXPR_SELECT:
1196            case EXPR_CONDITIONAL:
1197                    rl = handle_conditional_rl(expr, implied, recurse_cnt);
1198                    break;
1199            case EXPR_CALL:
1200                    rl = handle_call_rl(expr, implied, recurse_cnt);
1201                    break;
1202            case EXPR_STRING:
1203                    rl = NULL;
1204                    if (get_mtag_sval(expr, &sval))
1205                            rl = alloc_rl(sval, sval);
1206                    break;
1207            default:
1208                    rl = handle_variable(expr, implied, recurse_cnt);
1209            }

1211 out_cast:
1212            if (rl)
1213                    return rl;
1214            if (type && (implied == RL_ABSOLUTE || implied == RL_REAL_ABSOLUTE))
1215                    return alloc_whole_rl(type);
1216            return NULL;
1217 }

1219 struct {
1220            struct expression *expr;
1221            struct range_list *rl;
1222 } cached_results[24];
1223 static int cache_idx;

1225 void clear_math_cache(void)
1226 {
1227            memset(cached_results, 0, sizeof(cached_results));
1228 }

1230 /* returns 1 if it can get a value literal or else returns 0 */
1231 int get_value(struct expression *expr, sval_t *sval)
1232 {
1233            struct range_list *(*orig_custom_fn)(struct expression *expr);
1234            struct range_list *rl;
1235            int recurse_cnt = 0;
1236            sval_t tmp;
1237            int i;

1239            /*
1240             * This only handles RL_EXACT because other expr statements can be
1241             * different at different points.  Like the list iterator, for example.
1242             */
1243            for (i = 0; i < ARRAY_SIZE(cached_results); i++) {
1244                    if (expr == cached_results[i].expr)
1245                            return rl_to_sval(cached_results[i].rl, sval);
1246            }

1248            orig_custom_fn = custom_handle_variable;
```



```
1249            custom_handle_variable = NULL;
1250            rl = _get_rl(expr, RL_EXACT, &recurse_cnt);
1251            if (!rl_to_sval(rl, &tmp))
1252                    rl = NULL;
1253            custom_handle_variable = orig_custom_fn;

1255            cached_results[cache_idx].expr = expr;
1256            cached_results[cache_idx].rl = rl;
1257            cache_idx = (cache_idx + 1) % ARRAY_SIZE(cached_results);

1259            if (!rl)
1260                    return 0;

1262            *sval = tmp;
1263            return 1;
1264 }

1266 static int get_implied_value_internal(struct expression *expr, sval_t *sval, int
1267 {
1268            struct range_list *rl;

1270            rl =  _get_rl(expr, RL_IMPLIED, recurse_cnt);
1271            if (!rl_to_sval(rl, sval))
1272                    return 0;
1273            return 1;
1274 }

1276 int get_implied_value(struct expression *expr, sval_t *sval)
1277 {
1278            struct range_list *rl;
1279            int recurse_cnt = 0;

1281            rl =  _get_rl(expr, RL_IMPLIED, &recurse_cnt);
1282            if (!rl_to_sval(rl, sval))
1283                    return 0;
1284            return 1;
1285 }

1287 int get_implied_min(struct expression *expr, sval_t *sval)
1288 {
1289            struct range_list *rl;
1290            int recurse_cnt = 0;

1292            rl =  _get_rl(expr, RL_IMPLIED, &recurse_cnt);
1293            if (!rl)
1294                    return 0;
1295            *sval = rl_min(rl);
1296            return 1;
1297 }

1299 int get_implied_max(struct expression *expr, sval_t *sval)
1300 {
1301            struct range_list *rl;
1302            int recurse_cnt = 0;

1304            rl =  _get_rl(expr, RL_IMPLIED, &recurse_cnt);
1305            if (!rl)
1306                    return 0;
1307            *sval = rl_max(rl);
1308            return 1;
1309 }

1311 int get_implied_rl(struct expression *expr, struct range_list **rl)
1312 {
1313            int recurse_cnt = 0;
```

```
1315         *rl = _get_rl(expr, RL_IMPLIED, &recurse_cnt);
1316         if (*rl)
1317                 return 1;
1318         return 0;
1319 }

1321 static int get_absolute_rl_internal(struct expression *expr, struct range_list *
1322 {
1323         *rl = _get_rl(expr, RL_ABSOLUTE, recurse_cnt);
1324         if (!*rl)
1325                 *rl = alloc_whole_rl(get_type(expr));
1326         return 1;
1327 }

1329 int get_absolute_rl(struct expression *expr, struct range_list **rl)
1330 {
1331         int recurse_cnt = 0;

1333         *rl = _get_rl(expr, RL_ABSOLUTE, &recurse_cnt);
1334         if (!*rl)
1335                 *rl = alloc_whole_rl(get_type(expr));
1336         return 1;
1337 }

1339 int get_real_absolute_rl(struct expression *expr, struct range_list **rl)
1340 {
1341         int recurse_cnt = 0;

1343         *rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
1344         if (!*rl)
1345                 *rl = alloc_whole_rl(get_type(expr));
1346         return 1;
1347 }

1349 int custom_get_absolute_rl(struct expression *expr,
1350                            struct range_list *(*fn)(struct expression *expr),
1351                            struct range_list **rl)
1352 {
1353         int recurse_cnt = 0;

1355         *rl = NULL;
1356         custom_handle_variable = fn;
1357         *rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
1358         custom_handle_variable = NULL;
1359         return 1;
1360 }

1362 int get_implied_rl_var_sym(const char *var, struct symbol *sym, struct range_lis
1363 {
1364         struct smatch_state *state;

1366         state = get_state(SMATCH_EXTRA, var, sym);
1367         *rl = estate_rl(state);
1368         if (*rl)
1369                 return 1;
1370         return 0;
1371 }

1373 int get_hard_max(struct expression *expr, sval_t *sval)
1374 {
1375         struct range_list *rl;
1376         int recurse_cnt = 0;

1378         rl =  _get_rl(expr, RL_HARD, &recurse_cnt);
1379         if (!rl)
1380                 return 0;
```



```
1381         *sval = rl_max(rl);
1382         return 1;
1383 }

1385 int get_fuzzy_min(struct expression *expr, sval_t *sval)
1386 {
1387         struct range_list *rl;
1388         sval_t tmp;
1389         int recurse_cnt = 0;

1391         rl =  _get_rl(expr, RL_FUZZY, &recurse_cnt);
1392         if (!rl)
1393                 return 0;
1394         tmp = rl_min(rl);
1395         if (sval_is_negative(tmp) && sval_is_min(tmp))
1396                 return 0;
1397         *sval = tmp;
1398         return 1;
1399 }

1401 int get_fuzzy_max(struct expression *expr, sval_t *sval)
1402 {
1403         struct range_list *rl;
1404         sval_t max;
1405         int recurse_cnt = 0;

1407         rl =  _get_rl(expr, RL_FUZZY, &recurse_cnt);
1408         if (!rl)
1409                 return 0;
1410         max = rl_max(rl);
1411         if (max.uvalue > INT_MAX - 10000)
1412                 return 0;
1413         *sval = max;
1414         return 1;
1415 }

1417 int get_absolute_min(struct expression *expr, sval_t *sval)
1418 {
1419         struct range_list *rl;
1420         struct symbol *type;
1421         int recurse_cnt = 0;

1423         type = get_type(expr);
1424         if (!type)
1425                 type = &llong_ctype;  // FIXME: this is wrong but places assume
1426         rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
1427         if (rl)
1428                 *sval = rl_min(rl);
1429         else
1430                 *sval = sval_type_min(type);

1432         if (sval_cmp(*sval, sval_type_min(type)) < 0)
1433                 *sval = sval_type_min(type);
1434         return 1;
1435 }

1437 int get_absolute_max(struct expression *expr, sval_t *sval)
1438 {
1439         struct range_list *rl;
1440         struct symbol *type;
1441         int recurse_cnt = 0;

1443         type = get_type(expr);
1444         if (!type)
1445                 type = &llong_ctype;
1446         rl = _get_rl(expr, RL_REAL_ABSOLUTE, &recurse_cnt);
```

```
1447         if (rl)
1448                 *sval = rl_max(rl);
1449         else
1450                 *sval = sval_type_max(type);
1451
1452         if (sval_cmp(sval_type_max(type), *sval) < 0)
1453                 *sval = sval_type_max(type);
1454         return 1;
1455 }
1456
1457 int known_condition_true(struct expression *expr)
1458 {
1459         sval_t tmp;
1460
1461         if (!expr)
1462                 return 0;
1463
1464         if (get_value(expr, &tmp) && tmp.value)
1465                 return 1;
1466
1467         return 0;
1468 }
1469
1470 int known_condition_false(struct expression *expr)
1471 {
1472         if (!expr)
1473                 return 0;
1474
1475         if (is_zero(expr))
1476                 return 1;
1477
1478         return 0;
1479 }
1480
1481 int implied_condition_true(struct expression *expr)
1482 {
1483         sval_t tmp;
1484
1485         if (!expr)
1486                 return 0;
1487
1488         if (known_condition_true(expr))
1489                 return 1;
1490         if (get_implied_value(expr, &tmp) && tmp.value)
1491                 return 1;
1492
1493         if (expr->type == EXPR_POSTOP)
1494                 return implied_condition_true(expr->unop);
1495
1496         if (expr->type == EXPR_PREOP && expr->op == SPECIAL_DECREMENT)
1497                 return implied_not_equal(expr->unop, 1);
1498         if (expr->type == EXPR_PREOP && expr->op == SPECIAL_INCREMENT)
1499                 return implied_not_equal(expr->unop, -1);
1500
1501         expr = strip_expr(expr);
1502         switch (expr->type) {
1503         case EXPR_COMPARE:
1504                 if (do_comparison(expr) == 1)
1505                         return 1;
1506                 break;
1507         case EXPR_PREOP:
1508                 if (expr->op == '!') {
1509                         if (implied_condition_false(expr->unop))
1510                                 return 1;
1511                         break;
1512                 }
```



```
1513                 break;
1514         default:
1515                 if (implied_not_equal(expr, 0) == 1)
1516                         return 1;
1517                 break;
1518         }
1519         return 0;
1520 }
1521
1522 int implied_condition_false(struct expression *expr)
1523 {
1524         struct expression *tmp;
1525         sval_t sval;
1526
1527         if (!expr)
1528                 return 0;
1529
1530         if (known_condition_false(expr))
1531                 return 1;
1532
1533         switch (expr->type) {
1534         case EXPR_COMPARE:
1535                 if (do_comparison(expr) == 2)
1536                         return 1;
1537         case EXPR_PREOP:
1538                 if (expr->op == '!') {
1539                         if (implied_condition_true(expr->unop))
1540                                 return 1;
1541                         break;
1542                 }
1543                 tmp = strip_expr(expr);
1544                 if (tmp != expr)
1545                         return implied_condition_false(tmp);
1546                 break;
1547         default:
1548                 if (get_implied_value(expr, &sval) && sval.value == 0)
1549                         return 1;
1550                 break;
1551         }
1552         return 0;
1553 }
1554
1555 int can_integer_overflow(struct symbol *type, struct expression *expr)
1556 {
1557         int op;
1558         sval_t lmax, rmax, res;
1559
1560         if (!type)
1561                 type = &int_ctype;
1562
1563         expr = strip_expr(expr);
1564
1565         if (expr->type == EXPR_ASSIGNMENT) {
1566                 switch(expr->op) {
1567                 case SPECIAL_MUL_ASSIGN:
1568                         op = '*';
1569                         break;
1570                 case SPECIAL_ADD_ASSIGN:
1571                         op = '+';
1572                         break;
1573                 case SPECIAL_SHL_ASSIGN:
1574                         op = SPECIAL_LEFTSHIFT;
1575                         break;
1576                 default:
1577                         return 0;
1578                 }
```

```
1579         } else if (expr->type == EXPR_BINOP) {
1580                 if (expr->op != '*' && expr->op != '+' && expr->op != SPECIAL_LE
1581                         return 0;
1582                 op = expr->op;
1583         } else {
1584                 return 0;
1585         }

1587         get_absolute_max(expr->left, &lmax);
1588         get_absolute_max(expr->right, &rmax);

1590         if (sval_binop_overflows(lmax, op, rmax))
1591                 return 1;

1593         res = sval_binop(lmax, op, rmax);
1594         if (sval_cmp(res, sval_type_max(type)) > 0)
1595                 return 1;
1596         return 0;
1597 }
```

```
**********************************************************
    1270 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_mem_tracker.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2018 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include <unistd.h>

static int my_id;

static unsigned long max_size;

static void match_end_func(struct symbol *sym)
{
        FILE *file;
        char buf[1024];
        unsigned long size;

        file = fopen("/proc/self/statm", "r");
        if (!file)
                return;
        fread(buf, 1, sizeof(buf), file);
        fclose(file);

        size = strtoul(buf, NULL, 10);
        size = size * sysconf(_SC_PAGESIZE) / 1024;
        if (size > max_size)
                max_size = size;
}

unsigned long get_max_memory(void)
{
        return max_size;
}

void register_mem_tracker(int id)
{
        my_id = id;

        add_hook(&match_end_func, END_FUNC_HOOK);
}
```

```
**********************************************************
    7190 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_modification_hooks.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * There are a number of ways that variables are modified:
 * 1) assignment
 * 2) increment/decrement
 * 3) assembly
 * 4) inside functions.
 *
 * For setting stuff inside a function then, of course, it's more accurate if
 * you have the cross function database built.  Otherwise we are super
 * aggressive about marking things as modified and if you have "frob(foo);" then
 * we assume "foo->bar" is modified.
 */

#include <stdlib.h>
#include <stdio.h>
#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

enum {
        EARLY = 0,
        LATE = 1,
        BOTH = 2
};

static modification_hook **hooks;
static modification_hook **hooks_late;

ALLOCATOR(modification_data, "modification data");

static int my_id;
static struct smatch_state *alloc_my_state(struct expression *expr, struct smatc
{
        struct smatch_state *state;
        struct modification_data *data;
        char *name;

        state = __alloc_smatch_state(0);
        expr = strip_expr(expr);
        name = expr_to_str(expr);
        state->name = alloc_sname(name);
        free_string(name);
```



```
        data = __alloc_modification_data(0);
        data->prev = prev;
        data->cur = expr;
        state->data = data;

        return state;
}

void add_modification_hook(int owner, modification_hook *call_back)
{
        if (hooks[owner])
                sm_fatal("multiple modification hooks for %s", check_name(owner)
        hooks[owner] = call_back;
}

void add_modification_hook_late(int owner, modification_hook *call_back)
{
        if (hooks_late[owner])
                sm_fatal("multiple late modification hooks for %s", check_name(o
        hooks_late[owner] = call_back;
}

static int matches(char *name, struct symbol *sym, struct sm_state *sm)
{
        int len;

        if (sym != sm->sym)
                return false;

        len = strlen(name);
        if (strncmp(sm->name, name, len) == 0) {
                if (sm->name[len] == '\0')
                        return true;
                if (sm->name[len] == '-' || sm->name[len] == '.')
                        return true;
        }
        if (sm->name[0] != '*')
                return false;
        if (strncmp(sm->name + 1, name, len) == 0) {
                if (sm->name[len + 1] == '\0')
                        return true;
                if (sm->name[len + 1] == '-' || sm->name[len + 1] == '.')
                        return true;
        }
        return false;
}

static void call_modification_hooks_name_sym(char *name, struct symbol *sym, str
{
        struct sm_state *sm;
        struct smatch_state *prev;
        int match;

        prev = get_state(my_id, name, sym);

        if (cur_func_sym && !__in_fake_assign)
                set_state(my_id, name, sym, alloc_my_state(mod_expr, prev));

        FOR_EACH_SM(__get_cur_stree(), sm) {
                if (sm->owner > num_checks)
                        continue;
                match = matches(name, sym, sm);
                if (!match)
                        continue;

                if (late == EARLY || late == BOTH) {
```

```
127                         if (hooks[sm->owner])
128                                 (hooks[sm->owner])(sm, mod_expr);
129                 }
130                 if (late == LATE || late == BOTH) {
131                         if (hooks_late[sm->owner])
132                                 (hooks_late[sm->owner])(sm, mod_expr);
133                 }
134
135         } END_FOR_EACH_SM(sm);
136 }
137
138 static void call_modification_hooks(struct expression *expr, struct expression *
139 {
140         char *name;
141         struct symbol *sym;
142
143         if (late == LATE)
144                 update_mtag_data(expr);
145
146         name = expr_to_known_chunk_sym(expr, &sym);
147         if (!name)
148                 goto free;
149         call_modification_hooks_name_sym(name, sym, mod_expr, late);
150 free:
151         free_string(name);
152 }
153
154 static void db_param_add(struct expression *expr, int param, char *key, char *va
155 {
156         struct expression *arg, *gen_expr;
157         char *name, *other_name;
158         struct symbol *sym, *other_sym;
159
160         while (expr->type == EXPR_ASSIGNMENT)
161                 expr = strip_expr(expr->right);
162         if (expr->type != EXPR_CALL)
163                 return;
164
165         arg = get_argument_from_call_expr(expr->args, param);
166         if (!arg)
167                 return;
168
169         gen_expr = gen_expression_from_key(arg, key);
170         if (gen_expr)
171                 update_mtag_data(gen_expr);
172
173         name = get_variable_from_key(arg, key, &sym);
174         if (!name || !sym)
175                 goto free;
176
177         __in_fake_assign++;
178         call_modification_hooks_name_sym(name, sym, expr, BOTH);
179         __in_fake_assign--;
180
181         other_name = map_long_to_short_name_sym(name, sym, &other_sym);
182         if (other_name) {
183                 __in_fake_assign++;
184                 call_modification_hooks_name_sym(other_name, other_sym, expr, BO
185                 __in_fake_assign--;
186                 free_string(other_name);
187         }
188
189 free:
190         free_string(name);
191 }
```



```
193 static void match_assign(struct expression *expr, int late)
194 {
195         call_modification_hooks(expr->left, expr, late);
196 }
197
198 static void unop_expr(struct expression *expr, int late)
199 {
200         if (expr->op != SPECIAL_DECREMENT && expr->op != SPECIAL_INCREMENT)
201                 return;
202
203         call_modification_hooks(expr->unop, expr, late);
204 }
205
206 static void match_call(struct expression *expr)
207 {
208         struct expression *arg, *tmp;
209
210         /* If we have the DB then trust the DB */
211         if (!option_no_db)
212                 return;
213
214         FOR_EACH_PTR(expr->args, arg) {
215                 tmp = strip_expr(arg);
216                 if (tmp->type == EXPR_PREOP && tmp->op == '&')
217                         call_modification_hooks(tmp->unop, expr, BOTH);
218                 else
219                         call_modification_hooks(deref_expression(tmp), expr, BOT
220         } END_FOR_EACH_PTR(arg);
221 }
222
223 static void asm_expr(struct statement *stmt, int late)
224 {
225         struct expression *expr;
226         int state = 0;
227
228         FOR_EACH_PTR(stmt->asm_outputs, expr) {
229                 switch (state) {
230                 case 0: /* identifier */
231                 case 1: /* constraint */
232                         state++;
233                         continue;
234                 case 2: /* expression */
235                         state = 0;
236                         call_modification_hooks(expr, NULL, late);
237                         continue;
238                 }
239         } END_FOR_EACH_PTR(expr);
240 }
241
242
243 static void match_assign_early(struct expression *expr)
244 {
245         match_assign(expr, EARLY);
246 }
247
248 static void unop_expr_early(struct expression *expr)
249 {
250         unop_expr(expr, EARLY);
251 }
252
253 static void asm_expr_early(struct statement *stmt)
254 {
255         asm_expr(stmt, EARLY);
256 }
257
258 static void match_assign_late(struct expression *expr)
```

```
259 {
260         match_assign(expr, LATE);
261 }

263 static void unop_expr_late(struct expression *expr)
264 {
265         unop_expr(expr, LATE);
266 }

268 static void asm_expr_late(struct statement *stmt)
269 {
270         asm_expr(stmt, LATE);
271 }

273 struct smatch_state *get_modification_state(struct expression *expr)
274 {
275         return get_state_expr(my_id, expr);
276 }

278 void register_modification_hooks(int id)
279 {
280         my_id = id;

282         hooks = malloc((num_checks + 1) * sizeof(*hooks));
283         memset(hooks, 0, (num_checks + 1) * sizeof(*hooks));
284         hooks_late = malloc((num_checks + 1) * sizeof(*hooks));
285         memset(hooks_late, 0, (num_checks + 1) * sizeof(*hooks));

287         add_hook(&match_assign_early, ASSIGNMENT_HOOK);
288         add_hook(&unop_expr_early, OP_HOOK);
289         add_hook(&asm_expr_early, ASM_HOOK);
290 }

292 void register_modification_hooks_late(int id)
293 {
294         add_hook(&match_call, FUNCTION_CALL_HOOK);

296         select_return_states_hook(PARAM_ADD, &db_param_add);
297         select_return_states_hook(PARAM_SET, &db_param_add);

299         add_hook(&match_assign_late, ASSIGNMENT_HOOK_AFTER);
300         add_hook(&unop_expr_late, OP_HOOK);
301         add_hook(&asm_expr_late, ASM_HOOK);
302 }
```

```
**********************************************************
   12919 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_mtag.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.  All rights reserved.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * One problem that I have is that it's really hard to track how pointers are
 * passed around.  For example, it would be nice to know that the probe() and
 * remove() functions get the same pci_dev pointer.  It would be good to know
 * what pointers we're passing to the open() and close() functions.  But that
 * information gets lost in a call tree full of function pointer calls.
 *
 * I think the first step is to start naming specific pointers.  So when a
 * pointer is allocated, then it gets a tag.  So calls to kmalloc() generate a
 * tag.  But we might not use that, because there might be a better name like
 * framebuffer_alloc(). The framebuffer_alloc() is interesting because there is
 * one per driver and it's passed around to all the file operations.
 *
 * Perhaps we could make a list of functions like framebuffer_alloc() which take
 * a size and say that those are the interesting alloc functions.
 *
 * Another place where we would maybe name the pointer is when they are passed
 * to the probe().  Because that's an important pointer, since there is one
 * per driver (sort of).
 *
 * My vision is that you could take a pointer and trace it back to a global.  So
 * I'm going to track that pointer_tag - 28 bytes takes you to another pointer
 * tag.  You could follow that one back and so on.  Also when we pass a pointer
 * to a function that would be recorded as sort of a link or path or something.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

#include <openssl/md5.h>

static int my_id;

static struct smatch_state *alloc_tag_state(mtag_t tag)
{
        struct smatch_state *state;
        char buf[64];

        state = __alloc_smatch_state(0);
        snprintf(buf, sizeof(buf), "%lld", tag);
        state->name = alloc_sname(buf);
        state->data = malloc(sizeof(mtag_t));
        *(mtag_t *)state->data = tag;

        return state;
}

static mtag_t str_to_tag(const char *str)
{
        unsigned char c[MD5_DIGEST_LENGTH];
        unsigned long long *tag = (unsigned long long *)&c;
        MD5_CTX mdContext;
        int len;

        len = strlen(str);
        MD5_Init(&mdContext);
        MD5_Update(&mdContext, str, len);
        MD5_Final(c, &mdContext);

        *tag &= ~MTAG_ALIAS_BIT;
        *tag &= ~MTAG_OFFSET_MASK;

        return *tag;
}

static void alloc_assign(const char *fn, struct expression *expr, void *unused)
{
        struct expression *left, *right;
        char *left_name, *right_name;
        struct symbol *left_sym;
        char buf[256];
        mtag_t tag;


        // FIXME:  This should only happen when the size is not a paramter of
        // the caller
        return;

        if (expr->type != EXPR_ASSIGNMENT || expr->op != '=')
                return;
        left = strip_expr(expr->left);
        right = strip_expr(expr->right);
        if (right->type != EXPR_CALL || right->fn->type != EXPR_SYMBOL)
                return;

        left_name = expr_to_str_sym(left, &left_sym);
        right_name = expr_to_str(right);

        snprintf(buf, sizeof(buf), "%s %s %s %s", get_filename(), get_function()
                 left_name, right_name);
        tag = str_to_tag(buf);

        sql_insert_mtag_about(tag, left_name, right_name);

        if (left_name && left_sym)
                set_state(my_id, left_name, left_sym, alloc_tag_state(tag));

        free_string(left_name);
        free_string(right_name);
}

int get_string_mtag(struct expression *expr, mtag_t *tag)
{
        mtag_t xor;

        if (expr->type != EXPR_STRING || !expr->string)
                return 0;
```

```
128         /* I was worried about collisions so I added a xor */
129         xor = str_to_tag("__smatch string");
130         *tag = str_to_tag(expr->string->data);
131         *tag = *tag ^ xor;
132
133         return 1;
134 }
135
136 int get_toplevel_mtag(struct symbol *sym, mtag_t *tag)
137 {
138         char buf[256];
139
140         if (!sym)
141                 return 0;
142
143         if (!sym->ident ||
144             !(sym->ctype.modifiers & MOD_TOPLEVEL))
145                 return 0;
146
147         snprintf(buf, sizeof(buf), "%s %s",
148                  (sym->ctype.modifiers & MOD_STATIC) ? get_filename() : "extern"
149                  sym->ident->name);
150         *tag = str_to_tag(buf);
151         return 1;
152 }
153
154 int get_deref_mtag(struct expression *expr, mtag_t *tag)
155 {
156         mtag_t container_tag, member_tag;
157         int offset;
158
159         /*
160          * I'm not totally sure what I'm doing...
161          *
162          * This is supposed to get something like "global_var->ptr", but I don't
163          * feel like it's complete at all.
164          *
165          */
166
167         if (!get_mtag(expr->unop, &container_tag))
168                 return 0;
169
170         offset = get_member_offset_from_deref(expr);
171         if (offset < 0)
172                 return 0;
173
174         if (!mtag_map_select_tag(container_tag, -offset, &member_tag))
175                 return 0;
176
177         *tag = member_tag;
178         return 1;
179 }
180
181 static void global_variable(struct symbol *sym)
182 {
183         mtag_t tag;
184
185         if (!get_toplevel_mtag(sym, &tag))
186                 return;
187
188         sql_insert_mtag_about(tag,
189                               sym->ident->name,
190                               (sym->ctype.modifiers & MOD_STATIC) ? get_filename
191 }
```



```
193 static void db_returns_buf_size(struct expression *expr, int param, char *unused
194 {
195         struct expression *call;
196         struct range_list *rl;
197
198         if (expr->type != EXPR_ASSIGNMENT)
199                 return;
200         call = strip_expr(expr->right);
201
202         if (!parse_call_math_rl(call, math, &rl))
203                 return;
204 //      rl = cast_rl(&int_ctype, rl);
205 //      set_state_expr(my_size_id, expr->left, alloc_estate_rl(rl));
206 }
207
208 static void db_returns_memory_tag(struct expression *expr, int param, char *key,
209 {
210         struct expression *call, *arg;
211         mtag_t tag, alias;
212         char *name;
213         struct symbol *sym;
214
215         call = strip_expr(expr);
216         while (call->type == EXPR_ASSIGNMENT)
217                 call = strip_expr(call->right);
218         if (call->type != EXPR_CALL)
219                 return;
220
221         tag = strtoul(value, NULL, 10);
222
223         if (!create_mtag_alias(tag, call, &alias))
224                 return;
225
226         arg = get_argument_from_call_expr(call->args, param);
227         if (!arg)
228                 return;
229
230         name = get_variable_from_key(arg, key, &sym);
231         if (!name || !sym)
232                 goto free;
233
234         set_state(my_id, name, sym, alloc_tag_state(alias));
235 free:
236         free_string(name);
237 }
238
239 static void match_call_info(struct expression *expr)
240 {
241         struct smatch_state *state;
242         struct expression *arg;
243         int i = -1;
244
245         FOR_EACH_PTR(expr->args, arg) {
246                 i++;
247                 state = get_state_expr(my_id, arg);
248                 if (!state || !state->data)
249                         continue;
250                 sql_insert_caller_info(expr, MEMORY_TAG, i, "$", state->name);
251         } END_FOR_EACH_PTR(arg);
252 }
253
254 static void save_caller_info(const char *name, struct symbol *sym, char *key, ch
255 {
256         struct smatch_state *state;
257         char fullname[256];
258         mtag_t tag;
```

```
260         if (strncmp(key, "$", 1) != 0)
261                 return;

263         tag = atoll(value);
264         snprintf(fullname, 256, "%s%s", name, key + 1);
265         state = alloc_tag_state(tag);
266         set_state(my_id, fullname, sym, state);
267 }

269 static int get_array_mtag_offset(struct expression *expr, mtag_t *tag, int *offs
270 {
271         struct expression *array, *offset_expr;
272         struct symbol *type;
273         sval_t sval;

275         if (!is_array(expr))
276                 return 0;

278         array = get_array_base(expr);
279         if (!array)
280                 return 0;
281         type = get_type(array);
282         if (!type || type->type != SYM_ARRAY)
283                 return 0;
284         type = get_real_base_type(type);
285         if (!type_bytes(type))
286                 return 0;

288         if (!get_mtag(array, tag))
289                 return 0;

291         offset_expr = get_array_offset(expr);
292         if (!get_value(offset_expr, &sval))
293                 return 0;
294         *offset = sval.value * type_bytes(type);

296         return 1;
297 }

299 static int get_implied_mtag_offset(struct expression *expr, mtag_t *tag, int *of
300 {
301         struct smatch_state *state;
302         struct symbol *type;
303         sval_t sval;

305         type = get_type(expr);
306         if (!type_is_ptr(type))
307                 return 0;
308         state = get_extra_state(expr);
309         if (!state || !estate_get_single_value(state, &sval) || sval.value == 0)
310                 return 0;

312         *tag = sval.uvalue & ~MTAG_OFFSET_MASK;
313         *offset = sval.uvalue & MTAG_OFFSET_MASK;
314         return 1;
315 }

317 static int get_mtag_cnt;
318 int get_mtag(struct expression *expr, mtag_t *tag)
319 {
320         struct smatch_state *state;
321         int ret = 0;

323         expr = strip_expr(expr);
324         if (!expr)
```



```
325                 return 0;

327         if (get_mtag_cnt > 0)
328                 return 0;

330         get_mtag_cnt++;

332         switch (expr->type) {
333         case EXPR_STRING:
334                 if (get_string_mtag(expr, tag)) {
335                         ret = 1;
336                         goto dec_cnt;
337                 }
338                 break;
339         case EXPR_SYMBOL:
340                 if (get_toplevel_mtag(expr->symbol, tag)) {
341                         ret = 1;
342                         goto dec_cnt;
343                 }
344                 break;
345         case EXPR_DEREF:
346                 if (get_deref_mtag(expr, tag)) {
347                         ret = 1;
348                         goto dec_cnt;
349                 }
350                 break;
351         }

353         state = get_state_expr(my_id, expr);
354         if (!state)
355                 goto dec_cnt;
356         if (state->data) {
357                 *tag = *(mtag_t *)state->data;
358                 ret = 1;
359                 goto dec_cnt;
360         }

362 dec_cnt:
363         get_mtag_cnt--;
364         return ret;
365 }

367 int get_mtag_offset(struct expression *expr, mtag_t *tag, int *offset)
368 {
369         int val;

371         if (!expr)
372                 return 0;
373         if (expr->type == EXPR_PREOP && expr->op == '*')
374                 return get_mtag_offset(expr->unop, tag, offset);
375         if (get_implied_mtag_offset(expr, tag, offset))
376                 return 1;
377         if (!get_mtag(expr, tag))
378                 return 0;
379         expr = strip_expr(expr);
380         if (expr->type == EXPR_SYMBOL) {
381                 *offset = 0;
382                 return 1;
383         }
384         val = get_member_offset_from_deref(expr);
385         if (val < 0)
386                 return 0;
387         *offset = val;
388         return 1;
389 }
```

```
391 int create_mtag_alias(mtag_t tag, struct expression *expr, mtag_t *new)
392 {
393         char buf[256];
394         int lines_from_start;
395         char *str;
396
397         /*
398          * We need the alias to be unique.  It's not totally required that it
399          * be the same from one DB build to then next, but it makes debugging
400          * a bit simpler.
401          *
402          */
403
404         if (!cur_func_sym)
405                 return 0;
406
407         lines_from_start = expr->pos.line - cur_func_sym->pos.line;
408         str = expr_to_str(expr);
409         snprintf(buf, sizeof(buf), "%lld %d %s", tag, lines_from_start, str);
410         free_string(str);
411
412         *new = str_to_tag(buf);
413         sql_insert_mtag_alias(tag, *new);
414
415         return 1;
416 }
417
418 int expr_to_mtag_offset(struct expression *expr, mtag_t *tag, int *offset)
419 {
420         *offset = 0;
421
422         expr = strip_expr(expr);
423         if (!expr)
424                 return 0;
425
426         if (is_array(expr))
427                 return get_array_mtag_offset(expr, tag, offset);
428
429         if (expr->type ==  EXPR_DEREF) {
430                 *offset = get_member_offset_from_deref(expr);
431                 if (*offset < 0)
432                         return 0;
433                 return get_mtag(expr->deref, tag);
434         }
435
436         if (get_implied_mtag_offset(expr, tag, offset))
437                 return 1;
438
439         return get_mtag(expr, tag);
440 }
441
442 int get_mtag_sval(struct expression *expr, sval_t *sval)
443 {
444         struct symbol *type;
445         mtag_t tag;
446         int offset = 0;
447
448         if (bits_in_pointer != 64)
449                 return 0;
450
451         expr = strip_expr(expr);
452
453         type = get_type(expr);
454         if (!type_is_ptr(type))
455                 return 0;
456         /*
```



```
457          * There are only three options:
458          *
459          * 1) An array address:
460          *    p = array;
461          * 2) An address like so:
462          *    p = &my_struct->member;
463          * 3) A pointer:
464          *    p = pointer;
465          *
466          */
467
468         if (expr->type == EXPR_STRING && get_string_mtag(expr, &tag))
469                 goto found;
470
471         if (type->type == SYM_ARRAY && get_toplevel_mtag(expr->symbol, &tag))
472                 goto found;
473
474         if (get_implied_mtag_offset(expr, &tag, &offset))
475                 goto found;
476
477         if (expr->type != EXPR_PREOP || expr->op != '&')
478                 return 0;
479         expr = strip_expr(expr->unop);
480
481         if (!expr_to_mtag_offset(expr, &tag, &offset))
482                 return 0;
483         if (offset > MTAG_OFFSET_MASK)
484                 offset = MTAG_OFFSET_MASK;
485
486 found:
487         sval->type = type;
488         sval->uvalue = tag | offset;
489
490         return 1;
491 }
492
493 static struct expression *remove_dereference(struct expression *expr)
494 {
495         expr = strip_expr(expr);
496
497         if (expr->type == EXPR_PREOP && expr->op == '*')
498                 return strip_expr(expr->unop);
499         return preop_expression(expr, '&');
500 }
501
502 int get_mtag_addr_sval(struct expression *expr, sval_t *sval)
503 {
504         return get_mtag_sval(remove_dereference(expr), sval);
505 }
506
507 static void print_stored_to_mtag(int return_id, char *return_ranges, struct expr
508 {
509         struct sm_state *sm;
510         char buf[256];
511         const char *param_name;
512         int param;
513
514         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
515                 if (!sm->state->data)
516                         continue;
517
518                 param = get_param_num_from_sym(sm->sym);
519                 if (param < 0)
520                         continue;
521                 param_name = get_param_name(sm);
522                 if (!param_name)
```

```
523                     continue;
524             if (strcmp(param_name, "$") == 0)
525                     continue;

527             snprintf(buf, sizeof(buf), "%lld", *(mtag_t *)sm->state->data);
528             sql_insert_return_states(return_id, return_ranges, MEMORY_TAG, p
529     } END_FOR_EACH_SM(sm);
530 }

532 void register_mtag(int id)
533 {
534     my_id = id;


537     /*
538      * The mtag stuff only works on 64 systems because we store the
539      * information in the pointer itself.
540      * bit 63   : set for alias mtags
541      * bit 62-12: mtag hash
542      * bit 11-0 : offset
543      *
544      */
545     if (bits_in_pointer != 64)
546             return;

548     add_hook(&global_variable, BASE_HOOK);

550     add_function_assign_hook("kmalloc", &alloc_assign, NULL);
551     add_function_assign_hook("kzalloc", &alloc_assign, NULL);

553     select_return_states_hook(BUF_SIZE, &db_returns_buf_size);

555     add_hook(&match_call_info, FUNCTION_CALL_HOOK);
556     select_caller_info_hook(save_caller_info, MEMORY_TAG);
557     add_split_return_callback(&print_stored_to_mtag);
558     select_return_states_hook(MEMORY_TAG, db_returns_memory_tag);
559 }
```

```
**********************************************************
    5602 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_mtag_data.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2016 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * What we're doing here is saving all the possible values for static variables.
 * Later on we might do globals as well.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;
static struct stree *vals;

static int save_rl(void *_rl, int argc, char **argv, char **azColName)
{
        unsigned long *rl = _rl;

        *rl = strtoul(argv[0], NULL, 10);
        return 0;
}

static struct range_list *select_orig_rl(sval_t sval)
{
        struct range_list *rl = NULL;
        mtag_t tag = sval.uvalue & ~MTAG_OFFSET_MASK;
        int offset = sval.uvalue & MTAG_OFFSET_MASK;

        mem_sql(&save_rl, &rl, "select value from mtag_data where tag = %lld and
                tag, offset);
        return rl;
}

static int is_kernel_param(const char *name)
{
        struct sm_state *tmp;
        char buf[256];

        /*
         * I'm ignoring these because otherwise Smatch thinks that kernel
         * parameters are always set to the default.
         *
         */
```



```
        if (option_project != PROJ_KERNEL)
                return 0;

        snprintf(buf, sizeof(buf), "__param_%s.arg", name);

        FOR_EACH_SM(vals, tmp) {
                if (strcmp(tmp->name, buf) == 0)
                        return 1;
        } END_FOR_EACH_SM(tmp);

        return 0;
}

void insert_mtag_data(sval_t sval, struct range_list *rl)
{
        mtag_t tag = sval.uvalue & ~MTAG_OFFSET_MASK;
        int offset = sval.uvalue & MTAG_OFFSET_MASK;

        rl = clone_rl_permanent(rl);

        mem_sql(NULL, NULL, "delete from mtag_data where tag = %lld and offset =
                tag, offset, DATA_VALUE);
        mem_sql(NULL, NULL, "insert into mtag_data values (%lld, %d, %d, '%lu');
                tag, offset, DATA_VALUE, (unsigned long)rl);
}

void update_mtag_data(struct expression *expr)
{
        struct range_list *orig, *new, *rl;
        char *name;
        sval_t sval;

        name = expr_to_var(expr);
        if (is_kernel_param(name)) {
                free_string(name);
                return;
        }
        free_string(name);

        if (!get_mtag_addr_sval(expr, &sval))
                return;

        get_absolute_rl(expr, &rl);

        orig = select_orig_rl(sval);
        new = rl_union(orig, rl);
        insert_mtag_data(sval, new);
}

static void match_global_assign(struct expression *expr)
{
        struct range_list *rl;
        sval_t sval;
        char *name;

        name = expr_to_var(expr->left);
        if (is_kernel_param(name)) {
                free_string(name);
                return;
        }
        free_string(name);

        if (!get_mtag_addr_sval(expr->left, &sval))
                return;

        get_absolute_rl(expr->right, &rl);
```

```
127         insert_mtag_data(sval, rl);
128 }

130 static int save_mtag_data(void *_unused, int argc, char **argv, char **azColName
131 {
132         struct range_list *rl;

134         if (argc != 4) {
135                 sm_msg("Error saving mtag data");
136                 return 0;
137         }
138         if (!option_info)
139                 return 0;

141         rl = (struct range_list *)strtoul(argv[3], NULL, 10);
142         sm_msg("SQL: insert into mtag_data values ('%s', '%s', '%s', '%s');",
143                argv[0], argv[1], argv[2], show_rl(rl));

145         return 0;
146 }

148 static void match_end_file(struct symbol_list *sym_list)
149 {
150         mem_sql(&save_mtag_data, NULL, "select * from mtag_data where type = %d;
151                 DATA_VALUE);
152 }

154 struct db_info {
155         struct symbol *type;
156         struct range_list *rl;
157 };

159 static int get_vals(void *_db_info, int argc, char **argv, char **azColName)
160 {
161         struct db_info *db_info = _db_info;
162         struct range_list *tmp;

164         str_to_rl(db_info->type, argv[0], &tmp);
165         if (db_info->rl)
166                 db_info->rl = rl_union(db_info->rl, tmp);
167         else
168                 db_info->rl = tmp;

170         return 0;
171 }

173 struct db_cache_results {
174         sval_t sval;
175         struct range_list *rl;
176 };
177 static struct db_cache_results cached_results[8];

179 static int get_rl_from_mtag_sval(sval_t sval, struct symbol *type, struct range_
180 {
181         struct db_info db_info = {};
182         mtag_t tag;
183         int offset;
184         static int idx;
185         int ret;
186         int i;

188         for (i = 0; i < ARRAY_SIZE(cached_results); i++) {
189                 if (sval.uvalue == cached_results[i].sval.uvalue) {
190                         if (cached_results[i].rl) {
191                                 *rl = cached_results[i].rl;
192                                 return 1;
```



```
193                         }
194                         return 0;
195                 }
196         }

198         tag = sval.uvalue & ~MTAG_OFFSET_MASK;
199         offset = sval.uvalue & MTAG_OFFSET_MASK;
200         if (offset == MTAG_OFFSET_MASK) {
201                 ret = 0;
202                 goto update_cache;
203         }
204         db_info.type = type;

206         run_sql(get_vals, &db_info,
207                 "select value from mtag_data where tag = %lld and offset = %d an
208                 tag, offset, DATA_VALUE);
209         if (!db_info.rl || is_whole_rl(db_info.rl)) {
210                 db_info.rl = NULL;
211                 ret = 0;
212                 goto update_cache;
213         }

215         *rl = db_info.rl;
216         ret = 1;

218 update_cache:
219         cached_results[idx].sval = sval;
220         cached_results[idx].rl = db_info.rl;
221         idx = (idx + 1) % ARRAY_SIZE(cached_results);

223         return ret;
224 }

226 static void clear_cache(struct symbol *sym)
227 {
228         memset(cached_results, 0, sizeof(cached_results));
229 }

231 int get_mtag_rl(struct expression *expr, struct range_list **rl)
232 {
233         struct symbol *type;
234         sval_t sval;

236         if (!get_mtag_addr_sval(expr, &sval))
237                 return 0;

239         type = get_type(expr);
240         if (!type)
241                 return 0;

243         return get_rl_from_mtag_sval(sval, type, rl);
244 }

246 void register_mtag_data(int id)
247 {
248         my_id = id;

250         add_hook(&clear_cache, FUNC_DEF_HOOK);

252 //      if (!option_info)
253 //              return;
254         add_hook(&match_global_assign, GLOBAL_ASSIGNMENT_HOOK);
255         add_hook(&match_end_file, END_FILE_HOOK);
256 }
```

```
**********************************************************
    1523 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_mtag_map.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.  All rights reserved.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This basically stores when a pointer is stored as a struct member.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static void match_assign(struct expression *expr)
{
        struct expression *left, *right;
        mtag_t left_tag, right_tag;
        int offset;

        if (expr->op != '=')
                return;

        left = strip_expr(expr->left);
        right = strip_expr(expr->right);

        if (left->type != EXPR_DEREF)
                return;

        offset = get_member_offset_from_deref(left);
        if (offset < 0)
                return;

        if (!get_mtag(left->deref, &left_tag))
                return;
        if (!get_mtag(right, &right_tag))
                return;

        sql_insert_mtag_map(right_tag, -offset, left_tag);
}

void register_mtag_map(int id)
{
        my_id = id;

        add_hook(&match_assign, ASSIGNMENT_HOOK);
```



```
        add_hook(&match_assign, GLOBAL_ASSIGNMENT_HOOK);
}
```

```
**********************************************************
    6107 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_nul_terminator.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2018 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;
static int param_set_id;

STATE(terminated);
STATE(unterminated);
STATE(set);

static void set_terminated_var_sym(const char *name, struct symbol *sym, struct
{
        if (get_param_num_from_sym(sym) >= 0)
                set_state(param_set_id, name, sym, &set);
        set_state(my_id, name, sym, state);
}

static void set_terminated(struct expression *expr, struct smatch_state *state)
{
        struct symbol *sym;
        char *name;

        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                return;
        set_terminated_var_sym(name, sym, state);
        free_string(name);
}

static void match_nul_assign(struct expression *expr)
{
        struct expression *array;
        struct symbol *type;
        sval_t sval;

        if (expr->op != '=')
                return;

        if (!get_value(expr->right, &sval) || sval.value != 0)
                return;

        array = get_array_base(expr->left);
        if (!array)
```



```
                return;

        type = get_type(array);
        if (!type)
                return;
        type = get_real_base_type(type);
        if (type != &char_ctype)
                return;
        set_terminated(array, &terminated);
}

static struct smatch_state *get_terminated_state(struct expression *expr)
{
        struct sm_state *sm, *tmp;

        if (!expr)
                return NULL;
        if (expr->type == EXPR_STRING)
                return &terminated;
        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return NULL;
        if (sm->state == &terminated || sm->state == &unterminated)
                return sm->state;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (tmp->state == &unterminated)
                        return &unterminated;
        } END_FOR_EACH_PTR(tmp);

        return NULL;
}

static void match_string_assign(struct expression *expr)
{
        struct smatch_state *state;

        if (expr->op != '=')
                return;
        state = get_terminated_state(expr->right);
        if (!state)
                return;
        set_terminated(expr->left, state);
}

static int sm_to_term(struct sm_state *sm)
{
        struct sm_state *tmp;

        if (!sm)
                return -1;
        if (sm->state == &terminated)
                return 1;

        FOR_EACH_PTR(sm->possible, tmp) {
                if (tmp->state == &unterminated)
                        return 0;
        } END_FOR_EACH_PTR(tmp);

        return -1;
}

static void struct_member_callback(struct expression *call, int param, char *pri
{
        int term;
```

```
127         term = sm_to_term(sm);
128         if (term < 0)
129                 return;

131         sql_insert_caller_info(call, TERMINATED, param, printed_name, term ? "1"
132 }

134 static void match_call_info(struct expression *expr)
135 {
136         struct smatch_state *state;
137         struct expression *arg;
138         int i;

140         i = -1;
141         FOR_EACH_PTR(expr->args, arg) {
142                 i++;

144                 state = get_terminated_state(arg);
145                 if (!state)
146                         continue;
147                 sql_insert_caller_info(expr, TERMINATED, i, "$",
148                                        (state == &terminated) ? "1" : "0");
149         } END_FOR_EACH_PTR(arg);
150 }

152 static void caller_info_terminated(const char *name, struct symbol *sym, char *k
153 {
154         char fullname[256];

156         if (strcmp(key, "*$") == 0)
157                 snprintf(fullname, sizeof(fullname), "*%s", name);
158         else if (strncmp(key, "$", 1) == 0)
159                 snprintf(fullname, 256, "%s%s", name, key + 1);
160         else
161                 return;

163         set_state(my_id, fullname, sym, (*value == '1') ? &terminated : &untermi
164 }

166 static void split_return_info(int return_id, char *return_ranges, struct express
167 {
168         struct symbol *returned_sym;
169         struct sm_state *tmp, *sm;
170         const char *param_name;
171         int param;
172         int term;

174         FOR_EACH_MY_SM(param_set_id, __get_cur_stree(), tmp) {
175                 sm = get_sm_state(my_id, tmp->name, tmp->sym);
176                 if (!sm)
177                         continue;
178                 term = sm_to_term(sm);
179                 if (term < 0)
180                         continue;
181                 param = get_param_num_from_sym(tmp->sym);
182                 if (param < 0)
183                         continue;

185                 param_name = get_param_name(sm);
186                 if (!param_name)
187                         continue;
188                 if (strcmp(param_name, "$") == 0)
189                         continue;

191                 sql_insert_return_states(return_id, return_ranges, TERMINATED,
192                                          param, param_name, term ? "1" : "0");
```



```
193         } END_FOR_EACH_SM(tmp);

195         returned_sym = expr_to_sym(expr);
196         if (!returned_sym)
197                 return;

199         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
200                 if (sm->sym != returned_sym)
201                         continue;
202                 term = sm_to_term(sm);
203                 if (term < 0)
204                         continue;
205                 param_name = get_param_name(sm);
206                 if (!param_name)
207                         continue;
208                 sql_insert_return_states(return_id, return_ranges, TERMINATED,
209                                          -1, param_name, term ? "1" : "0");
210         } END_FOR_EACH_SM(sm);
211 }

213 static void return_info_terminated(struct expression *expr, int param, char *key
214 {
215         struct expression *arg;
216         char *name;
217         struct symbol *sym;

219         if (param == -1) {
220                 arg = expr->left;
221         } else {
222                 struct expression *call = expr;

224                 while (call->type == EXPR_ASSIGNMENT)
225                         call = strip_expr(call->right);
226                 if (call->type != EXPR_CALL)
227                         return;

229                 arg = get_argument_from_call_expr(call->args, param);
230                 if (!arg)
231                         return;
232         }

234         name = get_variable_from_key(arg, key, &sym);
235         if (!name || !sym)
236                 goto free;

238         set_terminated_var_sym(name, sym, (*value == '1') ? &terminated : &unter
239 free:
240         free_string(name);
241 }

243 bool is_nul_terminated(struct expression *expr)
244 {
245         if (get_terminated_state(expr) == &terminated)
246                 return 1;
247         return 0;
248 }

250 void register_nul_terminator(int id)
251 {
252         my_id = id;

254         add_hook(&match_nul_assign, ASSIGNMENT_HOOK);
255         add_hook(&match_string_assign, ASSIGNMENT_HOOK);

257         add_hook(&match_call_info, FUNCTION_CALL_HOOK);
258         add_member_info_callback(my_id, struct_member_callback);
```

```
259         add_split_return_callback(&split_return_info);

261         select_caller_info_hook(caller_info_terminated, TERMINATED);
262         select_return_states_hook(TERMINATED, return_info_terminated);
263 }

265 void register_nul_terminator_param_set(int id)
266 {
267         param_set_id = id;
268 }
```

```
**********************************************************
    5560 Fri Dec 21 15:00:27 2018
new/usr/src/tools/smatch/src/smatch_param_cleared.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This works together with smatch_clear_buffer.c.  This one is only for
 * tracking the information and smatch_clear_buffer.c changes SMATCH_EXTRA.
 *
 * This tracks functions like memset() which clear out a chunk of memory.
 * It fills in a gap that smatch_param_set.c can't handle.  It only handles
 * void pointers because smatch_param_set.c should handle the rest.  Oh.  And
 * also it handles arrays because Smatch sucks at handling arrays.
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

STATE(cleared);
STATE(zeroed);

static void db_param_cleared(struct expression *expr, int param, char *key, char
{
        struct expression *arg;
        char *name;
        struct symbol *sym;

        while (expr->type == EXPR_ASSIGNMENT)
                expr = strip_expr(expr->right);
        if (expr->type != EXPR_CALL)
                return;

        arg = get_argument_from_call_expr(expr->args, param);
        arg = strip_expr(arg);
        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        if (strcmp(value, "0") == 0)
                set_state(my_id, name, sym, &zeroed);
        else
                set_state(my_id, name, sym, &cleared);
free:
        free_string(name);
```



```
}

static void match_memset(const char *fn, struct expression *expr, void *arg)
{
        db_param_cleared(expr, PTR_INT(arg), (char *)"$", (char *)"0");
}

static void match_memcpy(const char *fn, struct expression *expr, void *arg)
{
        db_param_cleared(expr, PTR_INT(arg), (char *)"$", (char *)"");
}

static void print_return_value_param(int return_id, char *return_ranges, struct
{
        struct stree *stree;
        struct sm_state *sm;
        int param;
        const char *param_name;

        stree = __get_cur_stree();

        FOR_EACH_MY_SM(my_id, stree, sm) {
                param = get_param_num_from_sym(sm->sym);
                if (param < 0)
                        continue;

                param_name = get_param_name(sm);
                if (!param_name)
                        continue;

                if (sm->state == &zeroed) {
                        sql_insert_return_states(return_id, return_ranges,
                                                 PARAM_CLEARED, param, param_nam
                }

                if (sm->state == &cleared) {
                        sql_insert_return_states(return_id, return_ranges,
                                                 PARAM_CLEARED, param, param_nam
                }
        } END_FOR_EACH_SM(sm);
}

static void register_clears_param(void)
{
        struct token *token;
        char name[256];
        const char *function;
        int param;

        if (option_project == PROJ_NONE)
                return;

        snprintf(name, 256, "%s.clears_argument", option_project_str);

        token = get_tokens_file(name);
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                function = show_ident(token->ident);
                token = token->next;
                if (token_type(token) != TOKEN_NUMBER)
```

```
127                         return;
128                 param = atoi(token->number);
129                 add_function_hook(function, &match_memcpy, INT_PTR(param));
130                 token = token->next;
131         }
132         clear_token_alloc();
133 }

135 #define USB_DIR_IN 0x80
136 static void match_usb_control_msg(const char *fn, struct expression *expr, void
137 {
138         struct expression *inout;
139         sval_t sval;

141         inout = get_argument_from_call_expr(expr->args, 3);

143         if (get_value(inout, &sval) && !(sval.uvalue & USB_DIR_IN))
144                 return;

146         db_param_cleared(expr, 6, (char *)"$", (char *)"");
147 }

149 static void match_assign(struct expression *expr)
150 {
151         struct symbol *type;

153         /*
154          * If we have struct foo x, y; and we say that x = y; then it
155          * initializes the struct holes.  So we record that here.
156          */
157         type = get_type(expr->left);
158         if (!type || type->type != SYM_STRUCT)
159                 return;
160         set_state_expr(my_id, expr->left, &cleared);
161 }

163 static void match_array_assign(struct expression *expr)
164 {
165         struct expression *array_expr;

167         if (!is_array(expr->left))
168                 return;

170         array_expr = get_array_base(expr->left);
171         set_state_expr(my_id, array_expr, &cleared);
172 }

174 void register_param_cleared(int id)
175 {
176         my_id = id;

178         add_function_hook("memset", &match_memset, INT_PTR(0));
179         add_function_hook("memzero", &match_memset, INT_PTR(0));
180         add_function_hook("__memset", &match_memset, INT_PTR(0));
181         add_function_hook("__memzero", &match_memset, INT_PTR(0));

183         add_function_hook("memcpy", &match_memcpy, INT_PTR(0));
184         add_function_hook("memmove", &match_memcpy, INT_PTR(0));
185         add_function_hook("__memcpy", &match_memcpy, INT_PTR(0));
186         add_function_hook("__memmove", &match_memcpy, INT_PTR(0));
187         add_function_hook("strcpy", &match_memcpy, INT_PTR(0));
188         add_function_hook("strncpy", &match_memcpy, INT_PTR(0));
189         add_function_hook("sprintf", &match_memcpy, INT_PTR(0));
190         add_function_hook("snprintf", &match_memcpy, INT_PTR(0));

192         add_hook(&match_assign, ASSIGNMENT_HOOK);
```



```
193         add_hook(&match_array_assign, ASSIGNMENT_HOOK);

195         register_clears_param();

197         select_return_states_hook(PARAM_CLEARED, &db_param_cleared);
198         add_split_return_callback(&print_return_value_param);

200         if (option_project == PROJ_KERNEL) {
201                 add_function_hook("usb_control_msg", &match_usb_control_msg, NUL
202         }

204 }
```

```
**********************************************************
    9365 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_param_compare_limit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2016 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The point here is to store the relationships between two variables.
 * Ie:  y > x.
 * To do that we create a state with the two variables in alphabetical order:
 * ->name = "x vs y" and the state would be "<".  On the false path the state
 * would be ">=".
 *
 * Part of the trick of it is that if x or y is modified then we need to reset
 * the state.  We need to keep a list of all the states which depend on x and
 * all the states which depend on y.  The link_id code handles this.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int compare_id;
static int link_id;

static struct smatch_state *alloc_link_state(struct string_list *links)
{
        struct smatch_state *state;
        static char buf[256];
        char *tmp;
        int i;

        state = __alloc_smatch_state(0);

        i = 0;
        FOR_EACH_PTR(links, tmp) {
                if (!i++) {
                        snprintf(buf, sizeof(buf), "%s", tmp);
                } else {
                        append(buf, ", ", sizeof(buf));
                        append(buf, tmp, sizeof(buf));
                }
        } END_FOR_EACH_PTR(tmp);

        state->name = alloc_sname(buf);
        state->data = links;
        return state;
}
```



```
static struct smatch_state *merge_links(struct smatch_state *s1, struct smatch_s
{
        struct smatch_state *ret;
        struct string_list *links;

        links = combine_string_lists(s1->data, s2->data);
        ret = alloc_link_state(links);
        return ret;
}

static void save_link_var_sym(const char *var, struct symbol *sym, const char *l
{
        struct smatch_state *old_state, *new_state;
        struct string_list *links;
        char *new;

        old_state = get_state(link_id, var, sym);
        if (old_state)
                links = clone_str_list(old_state->data);
        else
                links = NULL;

        new = alloc_sname(link);
        insert_string(&links, new);

        new_state = alloc_link_state(links);
        set_state(link_id, var, sym, new_state);
}

static void add_comparison_var_sym(const char *left_name,
                struct var_sym_list *left_vsl,
                int comparison,
                const char *right_name, struct var_sym_list *right_vsl)
{
        struct smatch_state *state;
        struct var_sym *vs;
        char state_name[256];

        if (strcmp(left_name, right_name) > 0) {
                const char *tmp_name = left_name;
                struct var_sym_list *tmp_vsl = left_vsl;

                left_name = right_name;
                left_vsl = right_vsl;
                right_name = tmp_name;
                right_vsl = tmp_vsl;
                comparison = flip_comparison(comparison);
        }
        snprintf(state_name, sizeof(state_name), "%s vs %s", left_name, right_na
        state = alloc_compare_state(NULL, left_name, left_vsl, comparison, NULL,

        set_state(compare_id, state_name, NULL, state);

        FOR_EACH_PTR(left_vsl, vs) {
                save_link_var_sym(vs->var, vs->sym, state_name);
        } END_FOR_EACH_PTR(vs);
        FOR_EACH_PTR(right_vsl, vs) {
                save_link_var_sym(vs->var, vs->sym, state_name);
        } END_FOR_EACH_PTR(vs);
}

/*
 * This is quite a bit more limitted, less ambitious, simpler compared to
 * smatch_camparison.c.
 */
```

```
127 void __compare_param_limit_hook(struct expression *left_expr, struct expression
128                                 const char *state_name,
129                                 struct smatch_state *true_state, struct smatch_s
130 {
131         char *left_name = NULL;
132         char *right_name = NULL;
133         char *tmp_name = NULL;
134         struct symbol *left_sym, *right_sym, *tmp_sym;
135
136         left_name = expr_to_var_sym(left_expr, &left_sym);
137         if (!left_name || !left_sym)
138                 goto free;
139         right_name = expr_to_var_sym(right_expr, &right_sym);
140         if (!right_name || !right_sym)
141                 goto free;
142
143         if (get_param_num_from_sym(left_sym) < 0 ||
144             get_param_num_from_sym(right_sym) < 0)
145                 return;
146
147         tmp_name = get_other_name_sym(left_name, left_sym, &tmp_sym);
148         if (tmp_name) {
149                 free_string(left_name);
150                 left_name = tmp_name;
151                 left_sym = tmp_sym;
152         }
153
154         tmp_name = get_other_name_sym(right_name, right_sym, &tmp_sym);
155         if (tmp_name) {
156                 free_string(right_name);
157                 right_name = tmp_name;
158                 right_sym = tmp_sym;
159         }
160
161         if (param_was_set_var_sym(left_name, left_sym))
162                 return;
163         if (param_was_set_var_sym(right_name, right_sym))
164                 return;
165
166         set_true_false_states(compare_id, state_name, NULL, true_state, false_st
167         save_link_var_sym(left_name, left_sym, state_name);
168         save_link_var_sym(right_name, right_sym, state_name);
169 free:
170         free_string(left_name);
171         free_string(right_name);
172 }
173
174 static void print_return_comparison(int return_id, char *return_ranges, struct e
175 {
176         struct sm_state *tmp;
177         struct string_list *links;
178         char *link;
179         struct sm_state *sm;
180         struct compare_data *data;
181         struct var_sym *left, *right;
182         int left_param, right_param;
183         static char left_buf[256];
184         static char right_buf[256];
185         static char info_buf[256];
186         const char *tmp_name;
187
188         FOR_EACH_MY_SM(link_id, __get_cur_stree(), tmp) {
189                 links = tmp->state->data;
190                 FOR_EACH_PTR(links, link) {
191                         sm = get_sm_state(compare_id, link, NULL);
192                         if (!sm)
```



```
193                                 continue;
194                         data = sm->state->data;
195                         if (!data || !data->comparison)
196                                 continue;
197                         if (ptr_list_size((struct ptr_list *)data->left_vsl) !=
198                             ptr_list_size((struct ptr_list *)data->right_vsl) !=
199                                 continue;
200                         left = first_ptr_list((struct ptr_list *)data->left_vsl)
201                         right = first_ptr_list((struct ptr_list *)data->right_vs
202                         if (left->sym == right->sym &&
203                             strcmp(left->var, right->var) == 0)
204                                 continue;
205                         /*
206                          * Both parameters link to this comparison so only
207                          * record the first one.
208                          */
209                         if (left->sym != tmp->sym ||
210                             strcmp(left->var, tmp->name) != 0)
211                                 continue;
212
213                         left_param = get_param_num_from_sym(left->sym);
214                         right_param = get_param_num_from_sym(right->sym);
215                         if (left_param < 0 || right_param < 0) /* can't happen h
216                                 continue;
217
218                         tmp_name = get_param_name_var_sym(left->var, left->sym);
219                         if (!tmp_name)
220                                 continue;
221                         snprintf(left_buf, sizeof(left_buf), "%s", tmp_name);
222
223                         tmp_name = get_param_name_var_sym(right->var, right->sym
224                         if (!tmp_name || tmp_name[0] != '$')
225                                 continue;
226                         snprintf(right_buf, sizeof(right_buf), "$%d%s", right_pa
227
228                         snprintf(info_buf, sizeof(info_buf), "%s %s", show_speci
229                         sql_insert_return_states(return_id, return_ranges,
230                                                  COMPARE_LIMIT, left_param, left_buf, inf
231                 } END_FOR_EACH_PTR(link);
232
233         } END_FOR_EACH_SM(tmp);
234 }
235
236 static int parse_comparison(char **value, int *op)
237 {
238
239         *op = **value;
240
241         switch (*op) {
242         case '<':
243                 (*value)++;
244                 if (**value == '=') {
245                         (*value)++;
246                         *op = SPECIAL_LTE;
247                 }
248                 break;
249         case '=':
250                 (*value)++;
251                 (*value)++;
252                 *op = SPECIAL_EQUAL;
253                 break;
254         case '!':
255                 (*value)++;
256                 (*value)++;
257                 *op = SPECIAL_NOTEQUAL;
258                 break;
```

```
259         case '>':
260                 (*value)++;
261                 if (**value == '=') {
262                         (*value)++;
263                         *op = SPECIAL_GTE;
264                 }
265                 break;
266         default:
267                 return 0;
268         }
269
270         if (**value != ' ') {
271                 sm_perror("parsing comparison.  %s", *value);
272                 return 0;
273         }
274
275         (*value)++;
276         return 1;
277 }
278
279 static int split_op_param_key(char *value, int *op, int *param, char **key)
280 {
281         static char buf[256];
282         char *p;
283
284         if (!parse_comparison(&value, op))
285                 return 0;
286
287         snprintf(buf, sizeof(buf), value);
288
289         p = buf;
290         if (*p++ != '$')
291                 return 0;
292
293         *param = atoi(p);
294         if (*param < 0 || *param > 99)
295                 return 0;
296         p++;
297         if (*param > 9)
298                 p++;
299         p--;
300         *p = '$';
301         *key = p;
302
303         return 1;
304 }
305
306 static void db_return_comparison(struct expression *expr, int left_param, char *
307 {
308         struct expression *left_arg, *right_arg;
309         char *left_name = NULL;
310         struct symbol *left_sym;
311         char *right_name = NULL;
312         struct symbol *right_sym;
313         int op;
314         int right_param;
315         char *right_key;
316         struct var_sym_list *left_vsl = NULL, *right_vsl = NULL;
317
318         while (expr->type == EXPR_ASSIGNMENT)
319                 expr = strip_expr(expr->right);
320         if (expr->type != EXPR_CALL)
321                 return;
322
323         if (!split_op_param_key(value, &op, &right_param, &right_key))
324                 return;
```



```
325
326         left_arg = get_argument_from_call_expr(expr->args, left_param);
327         if (!left_arg)
328                 return;
329
330         right_arg = get_argument_from_call_expr(expr->args, right_param);
331         if (!right_arg)
332                 return;
333
334         left_name = get_variable_from_key(left_arg, key, &left_sym);
335         if (!left_name || !left_sym)
336                 goto free;
337         if (get_param_num_from_sym(left_sym) < 0)
338                 goto free;
339
340         right_name = get_variable_from_key(right_arg, right_key, &right_sym);
341         if (!right_name || !right_sym)
342                 goto free;
343         if (get_param_num_from_sym(right_sym) < 0)
344                 goto free;
345
346         add_var_sym(&left_vsl, left_name, left_sym);
347         add_var_sym(&right_vsl, right_name, right_sym);
348
349         add_comparison_var_sym(left_name, left_vsl, op, right_name, right_vsl);
350
351 free:
352         free_string(left_name);
353         free_string(right_name);
354 }
355
356 void register_param_compare_limit(int id)
357 {
358         compare_id = id;
359
360         add_merge_hook(compare_id, &merge_compare_states);
361         add_split_return_callback(&print_return_comparison);
362
363         select_return_states_hook(COMPARE_LIMIT, &db_return_comparison);
364 }
365
366 void register_param_compare_limit_links(int id)
367 {
368         link_id = id;
369
370         add_merge_hook(link_id, &merge_links);
371
372 }
```

```
**********************************************************
    5118 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_param_filter.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is for functions like:
 *
 * void foo(int *x)
 * {
 *      if (*x == 42)
 *              *x = 0;
 * }
 *
 * The final value of *x depends on the input to the function but with *x == 42
 * filtered out.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

static struct stree *start_states;
static struct stree_stack *saved_stack;
static void save_start_states(struct statement *stmt)
{
        start_states = get_all_states_stree(SMATCH_EXTRA);
}

static void free_start_states(void)
{
        free_stree(&start_states);
}

static void match_save_states(struct expression *expr)
{
        push_stree(&saved_stack, start_states);
        start_states = NULL;
}

static void match_restore_states(struct expression *expr)
{
        free_stree(&start_states);
        start_states = pop_stree(&saved_stack);
}
```



```
static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        struct smatch_state *state;

        if (parent_is_gone_var_sym(sm->name, sm->sym))
                return alloc_estate_empty();

        state = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (state)
                return state;
        return alloc_estate_whole(estate_type(sm->state));
}

static void pre_merge_hook(struct sm_state *sm)
{
        struct smatch_state *extra, *mine;
        struct range_list *rl;

        if (estate_rl(sm->state))
                return;

        extra = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (!extra)
                return;
        mine = get_state(my_id, sm->name, sm->sym);

        rl = rl_intersection(estate_rl(extra), estate_rl(mine));
        if (rl_equiv(rl, estate_rl(mine)))
                return;
        set_state(my_id, sm->name, sm->sym, alloc_estate_rl(clone_rl(rl)));
}

static void extra_mod_hook(const char *name, struct symbol *sym, struct expressi
{
        int param;

        if (__in_fake_assign)
                return;

        param = get_param_num_from_sym(sym);
        if (param < 0)
                return;

        /* on stack parameters are handled in smatch_param_limit.c */
        if (sym->ident && strcmp(sym->ident->name, name) == 0)
                return;

        set_state(my_id, name, sym, alloc_estate_empty());
}

/*
 * This relies on the fact that these states are stored so that
 * foo->bar is before foo->bar->baz.
 */
static int parent_set(struct string_list *list, const char *name)
{
        char *tmp;
        int len;
        int ret;

        FOR_EACH_PTR(list, tmp) {
                len = strlen(tmp);
                ret = strncmp(tmp, name, len);
                if (ret < 0)
                        continue;
```

```
127                 if (ret > 0)
128                         return 0;
129                 if (name[len] == '-')
130                         return 1;
131         } END_FOR_EACH_PTR(tmp);

133         return 0;
134 }

136 static void print_one_mod_param(int return_id, char *return_ranges,
137                         int param, struct sm_state *sm, struct string_list **tot
138 {
139         const char *param_name;

141         param_name = get_param_name(sm);
142         if (!param_name)
143                 return;
144         if (is_whole_rl(estate_rl(sm->state)))
145                 return;
146         if (!estate_rl(sm->state)) {
147                 insert_string(totally_filtered, (char *)sm->name);
148                 return;
149         }

151         sql_insert_return_states(return_id, return_ranges, PARAM_FILTER, param,
152                         param_name, show_rl(estate_rl(sm->state)));
153 }

155 static void print_return_value_param(int return_id, char *return_ranges, struct
156 {
157         struct sm_state *tmp;
158         struct sm_state *sm;
159         struct string_list *totally_filtered = NULL;
160         int param;

162         FOR_EACH_MY_SM(SMATCH_EXTRA, __get_cur_stree(), tmp) {
163                 param = get_param_num_from_sym(tmp->sym);
164                 if (param < 0)
165                         continue;

167                 /* on stack parameters are handled in smatch_param_limit.c */
168                 if (tmp->sym->ident && strcmp(tmp->sym->ident->name, tmp->name)
169                         continue;

171                 if (parent_set(totally_filtered, tmp->name))
172                         continue;

174                 sm = get_sm_state(my_id, tmp->name, tmp->sym);
175                 if (sm)
176                         print_one_mod_param(return_id, return_ranges, param, sm,
177         } END_FOR_EACH_SM(tmp);

179         free_ptr_list((struct ptr_list **)&totally_filtered);
180 }

182 int param_has_filter_data(struct sm_state *sm)
183 {
184         struct smatch_state *state;

186         state = get_state(my_id, sm->name, sm->sym);
187         if (!state) {
188                 if (get_assigned_expr_name_sym(sm->name, sm->sym))
189                         return 0;
190                 return 1;
191         }
192         if (estate_rl(state))
```



```
193                 return 1;
194         return 0;
195 }

197 void register_param_filter(int id)
198 {
199         my_id = id;

201         add_hook(&save_start_states, AFTER_DEF_HOOK);
202         add_hook(&free_start_states, AFTER_FUNC_HOOK);

204         add_extra_mod_hook(&extra_mod_hook);
205         add_unmatched_state_hook(my_id, &unmatched_state);
206         add_pre_merge_hook(my_id, &pre_merge_hook);
207         add_merge_hook(my_id, &merge_estates);

209         add_hook(&match_save_states, INLINE_FN_START);
210         add_hook(&match_restore_states, INLINE_FN_END);

212         add_split_return_callback(&print_return_value_param);
213 }
```

```
**********************************************************
    5649 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_param_limit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is almost the same as smatch_param_filter.c.  The difference is that
 * this only deals with values passed on the stack and param filter only deals
 * with values changed so that the caller sees the new value.  It other words
 * the key for these should always be "$" and the key for param_filter should
 * never be "$".  Also smatch_param_set() should never use "$" as the key.
 * Param set should work together with param_filter to determine the value that
 * the caller sees at the end.
 *
 * This is for functions like this:
 *
 * int foo(int a)
 * {
 *       if (a >= 0 && a < 10) {
 *               a = 42;
 *               return 1;
 *       }
 *       return 0;
 * }
 *
 * If we pass in 5, it returns 1.
 *
 * It's a bit complicated because we can't just consider the final value, we
 * have to always consider the passed in value.
 *
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

static struct stree *start_states;
static struct stree_stack *saved_stack;

static void save_start_states(struct statement *stmt)
{
        start_states = get_all_states_stree(SMATCH_EXTRA);
}

static void free_start_states(void)
```



```
{
        free_stree(&start_states);
}

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        struct smatch_state *state;

        state = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (state)
                return state;
        return alloc_estate_whole(estate_type(sm->state));
}

struct smatch_state *get_orig_estate(const char *name, struct symbol *sym)
{
        struct smatch_state *state;

        state = get_state(my_id, name, sym);
        if (state)
                return state;

        state = get_state(SMATCH_EXTRA, name, sym);
        if (state)
                return state;
        return alloc_estate_rl(alloc_whole_rl(get_real_base_type(sym)));
}

struct smatch_state *get_orig_estate_type(const char *name, struct symbol *sym,
{
        struct smatch_state *state;

        state = get_state(my_id, name, sym);
        if (state)
                return state;

        state = get_state(SMATCH_EXTRA, name, sym);
        if (state)
                return state;
        return alloc_estate_rl(alloc_whole_rl(type));
}

static struct range_list *generify_mtag_range(struct smatch_state *state)
{
        struct range_list *rl;
        struct data_range *drange;

        if (!estate_type(state) || estate_type(state)->type != SYM_PTR)
                return estate_rl(state);

        /*
         * The problem is that we get too specific on our param limits when we
         * know exactly what pointers are passed to a function.  It gets to the
         * point where we say "pointer x will succeed, but everything else will
         * fail."  And then we introduce a new caller which passes a different
         * pointer and it's like, "Sorry bro, that's not possible."
         *
         */
        rl = rl_intersection(estate_rl(state), valid_ptr_rl);
        if (!rl)
                return estate_rl(state);

        FOR_EACH_PTR(rl, drange) {
                if (drange->min.value != drange->max.value)
                        continue;
                if (drange->min.value > -4096 && drange->min.value <= 0)
```

```
127                     continue;
128             return rl_union(valid_ptr_rl, rl);
129         } END_FOR_EACH_PTR(drange);
130
131         return estate_rl(state);
132 }
133
134 static void print_return_value_param(int return_id, char *return_ranges, struct
135 {
136         struct smatch_state *state, *old;
137         struct sm_state *tmp;
138         struct range_list *rl;
139         const char *param_name;
140         int param;
141
142         FOR_EACH_MY_SM(SMATCH_EXTRA, __get_cur_stree(), tmp) {
143                 param = get_param_num_from_sym(tmp->sym);
144                 if (param < 0)
145                         continue;
146
147                 param_name = get_param_name(tmp);
148                 if (!param_name)
149                         continue;
150
151                 state = __get_state(my_id, tmp->name, tmp->sym);
152                 if (!state)
153                         state = tmp->state;
154
155                 if (estate_is_whole(state) || estate_is_empty(state))
156                         continue;
157                 old = get_state_stree(start_states, SMATCH_EXTRA, tmp->name, tmp
158                 if (old && rl_equiv(estate_rl(old), estate_rl(state)))
159                         continue;
160
161                 rl = generify_mtag_range(state);
162                 sql_insert_return_states(return_id, return_ranges, PARAM_LIMIT,
163                                          param, param_name, show_rl(rl));
164         } END_FOR_EACH_SM(tmp);
165 }
166
167 static void extra_mod_hook(const char *name, struct symbol *sym, struct expressi
168 {
169         struct smatch_state *orig_vals;
170         int param;
171
172         param = get_param_num_from_sym(sym);
173         if (param < 0)
174                 return;
175
176         orig_vals = get_orig_estate_type(name, sym, estate_type(state));
177         set_state(my_id, name, sym, orig_vals);
178 }
179
180 static void match_save_states(struct expression *expr)
181 {
182         push_stree(&saved_stack, start_states);
183         start_states = NULL;
184 }
185
186 static void match_restore_states(struct expression *expr)
187 {
188         free_stree(&start_states);
189         start_states = pop_stree(&saved_stack);
190 }
191
192 void register_param_limit(int id)
```



```
193 {
194         my_id = id;
195
196         add_hook(&save_start_states, AFTER_DEF_HOOK);
197         add_hook(&free_start_states, AFTER_FUNC_HOOK);
198
199         add_extra_mod_hook(&extra_mod_hook);
200         add_unmatched_state_hook(my_id, &unmatched_state);
201         add_merge_hook(my_id, &merge_estates);
202
203         add_hook(&match_save_states, INLINE_FN_START);
204         add_hook(&match_restore_states, INLINE_FN_END);
205
206         add_split_return_callback(&print_return_value_param);
207 }
```

```
**********************************************************
    6265 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_param_set.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is for functions like:
 *
 * int foo(int *x)
 * {
 *      if (*x == 42) {
 *              *x = 0;
 *              return 1;
 *      }
 *      return 0;
 * }
 *
 * If we return 1 that means the value of *x has been set to 0.  If we return
 * 0 then we have left *x alone.
 *
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static struct smatch_state *unmatched_state(struct sm_state *sm)
{
        return alloc_estate_empty();
}

static int parent_is_set(const char *name, struct symbol *sym, struct smatch_sta
{
        struct expression *faked;
        char *left_name;
        int ret = 0;
        int len;

        if (!__in_fake_assign)
                return 0;
        if (!is_whole_rl(estate_rl(state)))
                return 0;
        if (get_state(my_id, name, sym))
                return 0;
```



```
        faked = get_faked_expression();
        if (!faked)
                return 0;
        if ((faked->type == EXPR_PREOP || faked->type == EXPR_POSTOP) &&
            (faked->op == SPECIAL_INCREMENT || faked->op == SPECIAL_DECREMENT))
                faked = strip_expr(faked->unop);
                if (faked->type == EXPR_SYMBOL)
                        return 1;
                return 0;
        }
        if (faked->type != EXPR_ASSIGNMENT)
                return 0;

        left_name = expr_to_var(faked->left);
        if (!left_name)
                return 0;

        len = strlen(left_name);
        if (strncmp(name, left_name, len) == 0 && name[len] == '-')
                ret = 1;
        free_string(left_name);

        return ret;
}

static void extra_mod_hook(const char *name, struct symbol *sym, struct expressi
{
        if (parent_is_set(name, sym, state))
                return;
        if (get_param_num_from_sym(sym) < 0)
                return;
        set_state(my_id, name, sym, state);
}

/*
 * This function is is a dirty hack because extra_mod_hook is giving us a NULL
 *  sym instead of a vsl.
 */
static void match_array_assignment(struct expression *expr)
{
        struct expression *array, *offset;
        char *name;
        struct symbol *sym;
        struct range_list *rl;
        sval_t sval;
        char buf[256];

        if (__in_fake_assign)
                return;

        if (!is_array(expr->left))
                return;
        array = get_array_base(expr->left);
        offset = get_array_offset(expr->left);

        /* These are handled by extra_mod_hook() */
        if (get_value(offset, &sval))
                return;
        name = expr_to_var_sym(array, &sym);
        if (!name || !sym)
                goto free;
        if (get_param_num_from_sym(sym) < 0)
                goto free;
        get_absolute_rl(expr->right, &rl);
        rl = cast_rl(get_type(expr->left), rl);
```

```
127         snprintf(buf, sizeof(buf), "*%s", name);
128         set_state(my_id, buf, sym, alloc_estate_rl(rl));
129 free:
130         free_string(name);
131 }

133 /*
134  * This relies on the fact that these states are stored so that
135  * foo->bar is before foo->bar->baz.
136  */
137 static int parent_set(struct string_list *list, const char *name)
138 {
139         char *tmp;
140         int len;
141         int ret;

143         FOR_EACH_PTR(list, tmp) {
144                 len = strlen(tmp);
145                 ret = strncmp(tmp, name, len);
146                 if (ret < 0)
147                         continue;
148                 if (ret > 0)
149                         return 0;
150                 if (name[len] == '-')
151                         return 1;
152         } END_FOR_EACH_PTR(tmp);

154         return 0;
155 }

157 static void print_return_value_param(int return_id, char *return_ranges, struct
158 {
159         struct sm_state *sm;
160         struct smatch_state *extra;
161         int param;
162         struct range_list *rl;
163         const char *param_name;
164         struct string_list *set_list = NULL;
165         char *math_str;
166         char buf[256];
167         sval_t sval;

169         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
170                 if (!estate_rl(sm->state))
171                         continue;
172                 extra = get_state(SMATCH_EXTRA, sm->name, sm->sym);
173                 if (extra) {
174                         rl = rl_intersection(estate_rl(sm->state), estate_rl(ext
175                         if (!rl)
176                                 continue;
177                 } else {
178                         rl = estate_rl(sm->state);
179                 }

181                 param = get_param_num_from_sym(sm->sym);
182                 if (param < 0)
183                         continue;
184                 param_name = get_param_name(sm);
185                 if (!param_name)
186                         continue;
187                 if (strcmp(param_name, "$") == 0) {
188                         insert_string(&set_list, (char *)sm->name);
189                         continue;
190                 }

192                 if (rl_to_sval(rl, &sval)) {
```



```
193                         insert_string(&set_list, (char *)sm->name);
194                         sql_insert_return_states(return_id, return_ranges,
195                                         param_has_filter_data(sm) ? PARAM_ADD :
196                                         param, param_name, show_rl(rl));
197                         continue;
198                 }

200                 math_str = get_value_in_terms_of_parameter_math_var_sym(sm->name
201                 if (math_str) {
202                         snprintf(buf, sizeof(buf), "%s[%s]", show_rl(rl), math_s
203                         insert_string(&set_list, (char *)sm->name);
204                         sql_insert_return_states(return_id, return_ranges,
205                                         param_has_filter_data(sm) ? PARAM_ADD :
206                                         param, param_name, buf);
207                         continue;
208                 }

210                 /* no useful information here. */
211                 if (is_whole_rl(rl) && parent_set(set_list, sm->name))
212                         continue;
213                 insert_string(&set_list, (char *)sm->name);

215                 sql_insert_return_states(return_id, return_ranges,
216                                          param_has_filter_data(sm) ? PARAM_ADD :
217                                          param, param_name, show_rl(rl));

219         } END_FOR_EACH_SM(sm);

221         free_ptr_list((struct ptr_list **)&set_list);
222 }

224 int param_was_set_var_sym(const char *name, struct symbol *sym)
225 {
226         struct sm_state *sm;
227         int len;

229         FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
230                 if (sm->sym != sym)
231                         continue;
232                 len = strlen(sm->name);
233                 if (strncmp(sm->name, name, len) != 0)
234                         continue;
235                 if (name[len] == '\0' ||
236                     name[len] == '-')
237                         return 1;
238         } END_FOR_EACH_SM(sm);

240         return 0;
241 }

243 int param_was_set(struct expression *expr)
244 {
245         char *name;
246         struct symbol *sym;
247         int ret = 0;

249         name = expr_to_var_sym(expr, &sym);
250         if (!name || !sym)
251                 goto free;

253         ret = param_was_set_var_sym(name, sym);
254 free:
255         free_string(name);
256         return ret;
257 }
```

```
259 void register_param_set(int id)
260 {
261         my_id = id;

263         add_extra_mod_hook(&extra_mod_hook);
264         add_hook(match_array_assignment, ASSIGNMENT_HOOK);
265         add_unmatched_state_hook(my_id, &unmatched_state);
266         add_merge_hook(my_id, &merge_estates);
267         add_split_return_callback(&print_return_value_param);
268 }
```

```
**********************************************************
    5933 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_param_to_mtag_data.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Take a look at request_threaded_irq().  It takes thread_fn and dev_id.  Then
 * it does:
 *
 *      action = kzalloc(sizeof(struct irqaction), GFP_KERNEL);
 *      action->thread_fn = thread_fn;
 *      action->dev_id = dev_id;
 *
 * It doesn't ever pass action back to the higher levels, but instead registers
 * it with the lower levels.
 *
 * The kzalloc() allocation creates a new mtag.  We don't know at this point
 * what "thread_fn" and "dev_id" are because they come from many different
 * sources.
 *
 * So what we do is we pass the information back to the callers that thread_fn
 * and dev_id are stored as a specific mtag data.  Then when the callers *do*
 * know what values are passed they create an mtag_alias.  An mtag_alias is a
 * many to one relationship.  Then they store that in mtag_data using the
 * mtag_alias.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

static int my_id;

struct tag_assign_info {
        mtag_t tag;
        int offset;
};
ALLOCATOR(tag_assign_info, "tag name offset");

static struct smatch_state *alloc_tag_data_state(mtag_t tag, char *name, int off
{
        struct smatch_state *state;
        struct tag_assign_info *data;

        data = __alloc_tag_assign_info(0);
        data->tag = tag;
        data->offset = offset;
```



```
        state = __alloc_smatch_state(0);
        state->name = alloc_sname(name);
        state->data = data;
        return state;
}

struct smatch_state *merge_tag_info(struct smatch_state *s1, struct smatch_state
{
        /* Basically ignore undefined states */
        if (s1 == &undefined)
                return s2;
        if (s2 == &undefined)
                return s1;

        return &merged;
}

static void match_assign(struct expression *expr)
{
        struct expression *left;
        struct symbol *right_sym;
        char *name;
        mtag_t tag;
        int offset;
        int param;

        if (expr->op != '=')
                return;
        left = strip_expr(expr->left);
        right_sym = expr_to_sym(expr->right);
        if (!right_sym)
                return;

        param = get_param_num_from_sym(right_sym);
        if (param < 0)
                return;
        // FIXME:  modify param_has_filter_data() to take a name/sym
        if (!expr_to_mtag_offset(left, &tag, &offset))
                return;
        name = expr_to_str(left);
        if (!name)
                return;
        set_state_expr(my_id, expr->right, alloc_tag_data_state(tag, name, offse
        free_string(name);
}

#if 0
static void save_mtag_to_map(struct expression *expr, mtag_t tag, int offset, in
{
        struct expression *arg, *gen_expr;
        mtag_t arg_tag;

        arg = get_argument_from_call_expr(expr->args, param);
        if (!arg)
                return;

        gen_expr = gen_expression_from_key(arg, key);
        if (!gen_expr)
                return;

        if (!get_mtag(gen_expr, &arg_tag))
                arg_tag = 0;

        if (local_debug)
                sm_msg("finding mtag for '%s' %lld", expr_to_str(gen_expr), arg_
```

```
}
#endif

static void propogate_assignment(struct expression *expr, mtag_t tag, int offset
{
        struct expression *arg;
        int orig_param;
        char buf[32];
        char *name;
        struct symbol *sym;

        arg = get_argument_from_call_expr(expr->args, param);
        if (!arg)
                return;
        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        orig_param = get_param_num_from_sym(sym);
        if (orig_param < 0)
                goto free;

        snprintf(buf, sizeof(buf), "$->[%d]", offset);
        set_state(my_id, name, sym, alloc_tag_data_state(tag, buf, offset));
free:
        free_string(name);
}

static void assign_to_alias(struct expression *expr, int param, mtag_t tag, int
{
        struct expression *arg, *gen_expr;
        struct range_list *rl;
        mtag_t arg_tag;
        mtag_t alias;

        arg = get_argument_from_call_expr(expr->args, param);
        if (!arg)
                return;

        gen_expr = gen_expression_from_key(arg, key);
        if (!gen_expr)
                return;

        get_absolute_rl(gen_expr, &rl);

        if (!create_mtag_alias(tag, expr, &alias))
                return;

//      insert_mtag_data(alias, offset, rl);

        if (get_mtag(gen_expr, &arg_tag))
                sql_insert_mtag_map(arg_tag, -offset, alias);
}

static void call_does_mtag_assign(struct expression *expr, int param, char *key,
{
        char *p;
        mtag_t tag;
        int offset;

        while (expr->type == EXPR_ASSIGNMENT)
                expr = strip_expr(expr->right);
        if (expr->type != EXPR_CALL)
                return;

        tag = strtoul(value, NULL, 10);
```



```
        p = strchr(value, '+');
        if (!p)
                return;
        offset = atoi(p + 1);

//      save_mtag_to_map(expr, tag, offset, param, key, value);
        propogate_assignment(expr, tag, offset, param, key);
        assign_to_alias(expr, param, tag, offset, key);
}

static void print_stored_to_mtag(int return_id, char *return_ranges, struct expr
{
        struct sm_state *sm;
        struct tag_assign_info *data;
        char buf[256];
        const char *param_name;
        int param;

        FOR_EACH_MY_SM(my_id, __get_cur_stree(), sm) {
                if (!sm->state->data)
                        continue;

                param = get_param_num_from_sym(sm->sym);
                if (param < 0)
                        continue;
                param_name = get_param_name(sm);
                if (!param_name)
                        continue;

                data = sm->state->data;
                snprintf(buf, sizeof(buf), "%lld+%d", data->tag, data->offset);
                sql_insert_return_states(return_id, return_ranges, MTAG_ASSIGN,
        } END_FOR_EACH_SM(sm);
}

void register_param_to_mtag_data(int id)
{
        my_id = id;

        add_hook(&match_assign, ASSIGNMENT_HOOK);
        select_return_states_hook(MTAG_ASSIGN, &call_does_mtag_assign);
        add_merge_hook(my_id, &merge_tag_info);
        add_split_return_callback(&print_stored_to_mtag);
}
```

```
**********************************************************
    2486 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_param_used.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static struct stree *used_stree;
static struct stree_stack *saved_stack;

STATE(used);

static void get_state_hook(int owner, const char *name, struct symbol *sym)
{
        int arg;

        if (!option_info)
                return;
        if (__in_fake_assign)
                return;

        arg = get_param_num_from_sym(sym);
        if (arg >= 0)
                set_state_stree(&used_stree, my_id, name, sym, &used);
}

static void set_param_used(struct expression *call, struct expression *arg, char
{
        struct symbol *sym;
        char *name;
        int arg_nr;

        name = get_variable_from_key(arg, key, &sym);
        if (!name || !sym)
                goto free;

        arg_nr = get_param_num_from_sym(sym);
        if (arg_nr >= 0)
                set_state(my_id, name, sym, &used);
free:
        free_string(name);
}

static void process_states(void)
{
```



```
        struct sm_state *tmp;
        int arg;
        const char *name;

        FOR_EACH_SM(used_stree, tmp) {
                arg = get_param_num_from_sym(tmp->sym);
                if (arg < 0)
                        continue;
                name = get_param_name(tmp);
                if (!name)
                        continue;

                sql_insert_return_implies(PARAM_USED, arg, name, "");
        } END_FOR_EACH_SM(tmp);

        free_stree(&used_stree);
}

static void match_function_def(struct symbol *sym)
{
        free_stree(&used_stree);
}

static void match_save_states(struct expression *expr)
{
        push_stree(&saved_stack, used_stree);
        used_stree = NULL;
}

static void match_restore_states(struct expression *expr)
{
        free_stree(&used_stree);
        used_stree = pop_stree(&saved_stack);
}

void register_param_used(int id)
{
        my_id = id;

        add_hook(&match_function_def, FUNC_DEF_HOOK);

        add_get_state_hook(&get_state_hook);

        add_hook(&match_save_states, INLINE_FN_START);
        add_hook(&match_restore_states, INLINE_FN_END);

        select_return_implies_hook(PARAM_USED, &set_param_used);
        all_return_states_hook(&process_states);
}
```

**********
*1124 Fri Dec 21 15:00:28 2018*
*new/usr/src/tools/smatch/src/smatch_parameter_names.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static void match_def(struct symbol *sym)
{
        struct symbol *param;
        int i;

        if (__inline_fn)
                return;

        i = -1;
        FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, param) {
                i++;
                if (!param->ident)
                        continue;
                sql_insert_parameter_name(i, param->ident->name);
        } END_FOR_EACH_PTR(param);
}

void register_parameter_names(int id)
{
        if (!option_info)
                return;

        add_hook(&match_def, FUNC_DEF_HOOK);
}
```

```
**********************************************************
   12996 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_parse_call_math.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

struct {
        const char *func;
        int param;
} alloc_functions[] = {
        {"kmalloc", 0},
        {"kzalloc", 0},
        {"__kmalloc", 0},
        {"vmalloc", 0},
        {"__vmalloc", 0},
        {"__vmalloc_node", 0},
};

static struct range_list_stack *rl_stack;
static struct string_list *op_list;

static void push_op(char c)
{
        char *p;

        p = malloc(1);
        p[0] = c;
        add_ptr_list(&op_list, p);
}

static char pop_op(void)
{
        char *p;
        char c;

        if (!op_list) {
                sm_perror("%s: no op_list", __func__);
                return '\0';
        }

        p = last_ptr_list((struct ptr_list *)op_list);

        delete_ptr_list_last((struct ptr_list **)&op_list);
```



```
        c = p[0];
        free(p);

        return c;
}

static int op_precedence(char c)
{
        switch (c) {
        case '+':
        case '-':
                return 1;
        case '*':
        case '/':
                return 2;
        default:
                return 0;
        }
}

static int top_op_precedence(void)
{
        char *p;

        if (!op_list)
                return 0;

        p = last_ptr_list((struct ptr_list *)op_list);
        return op_precedence(p[0]);
}

static void rl_pop_until(char c)
{
        char op;
        struct range_list *left, *right;
        struct range_list *res;

        while (top_op_precedence() && op_precedence(c) <= top_op_precedence()) {
                op = pop_op();
                right = pop_rl(&rl_stack);
                left = pop_rl(&rl_stack);
                res = rl_binop(left, op, right);
                if (!res)
                        res = alloc_whole_rl(&llong_ctype);
                push_rl(&rl_stack, res);
        }
}

static void rl_discard_stacks(void)
{
        while (op_list)
                pop_op();
        while (rl_stack)
                pop_rl(&rl_stack);
}

static int read_rl_from_var(struct expression *call, char *p, char **end, struct
{
        struct expression *arg;
        struct smatch_state *state;
        long param;
        char *name;
        struct symbol *sym;
        char buf[256];
        int star;
```

```
127         p++;
128         param = strtol(p, &p, 10);

130         arg = get_argument_from_call_expr(call->args, param);
131         if (!arg)
132                 return 0;

134         if (*p != '-' && *p != '.') {
135                 get_absolute_rl(arg, rl);
136                 *end = p;
137                 return 1;
138         }

140         *end = strchr(p, ' ');

142         if (arg->type == EXPR_PREOP && arg->op == '&') {
143                 arg = strip_expr(arg->unop);
144                 star = 0;
145                 p++;
146         } else {
147                 star = 1;
148                 p += 2;
149         }

151         name = expr_to_var_sym(arg, &sym);
152         if (!name)
153                 return 0;
154         snprintf(buf, sizeof(buf), "%s%s", name, star ? "->" : ".");
155         free_string(name);

157         if (*end - p + strlen(buf) >= sizeof(buf))
158                 return 0;
159         strncat(buf, p, *end - p);

161         state = get_state(SMATCH_EXTRA, buf, sym);
162         if (!state)
163                 return 0;
164         *rl = estate_rl(state);
165         return 1;
166 }

168 static int read_var_num(struct expression *call, char *p, char **end, struct ran
169 {
170         sval_t sval;

172         while (*p == ' ')
173                 p++;

175         if (*p == '$')
176                 return read_rl_from_var(call, p, end, rl);

178         sval.type = &llong_ctype;
179         sval.value = strtoll(p, end, 10);
180         if (*end == p)
181                 return 0;
182         *rl = alloc_rl(sval, sval);
183         return 1;
184 }

186 static char *read_op(char *p)
187 {
188         while (*p == ' ')
189                 p++;

191         switch (*p) {
192         case '+':
```



```
193         case '-':
194         case '*':
195         case '/':
196                 return p;
197         default:
198                 return NULL;
199         }
200 }

202 int parse_call_math_rl(struct expression *call, char *math, struct range_list **
203 {
204         struct range_list *tmp;
205         char *c;

207         /* try to implement shunting yard algorithm. */

209         c = (char *)math;
210         while (1) {
211                 if (option_debug)
212                         sm_msg("parsing %s", c);

214                 /* read a number and push it onto the number stack */
215                 if (!read_var_num(call, c, &c, &tmp))
216                         goto fail;
217                 push_rl(&rl_stack, tmp);

219                 if (option_debug)
220                         sm_msg("val = %s remaining = %s", show_rl(tmp), c);

222                 if (!*c)
223                         break;
224                 if (*c == ']' && *(c + 1) == '\0')
225                         break;

227                 c = read_op(c);
228                 if (!c)
229                         goto fail;

231                 if (option_debug)
232                         sm_msg("op = %c remaining = %s", *c, c);

234                 rl_pop_until(*c);
235                 push_op(*c);
236                 c++;
237         }

239         rl_pop_until(0);
240         *rl = pop_rl(&rl_stack);
241         return 1;
242 fail:
243         rl_discard_stacks();
244         return 0;
245 }

247 int parse_call_math(struct expression *call, char *math, sval_t *sval)
248 {
249         struct range_list *rl;

251         if (!parse_call_math_rl(call, math, &rl))
252                 return 0;
253         if (!rl_to_sval(rl, sval))
254                 return 0;
255         return 1;
256 }

258 static struct smatch_state *alloc_state_sname(char *sname)
```

```
259 {
260         struct smatch_state *state;
261
262         state = __alloc_smatch_state(0);
263         state->name = sname;
264         state->data = INT_PTR(1);
265         return state;
266 }
267
268 static int get_arg_number(struct expression *expr)
269 {
270         struct symbol *sym;
271         struct symbol *arg;
272         int i;
273
274         expr = strip_expr(expr);
275         if (expr->type != EXPR_SYMBOL)
276                 return -1;
277         sym = expr->symbol;
278
279         i = 0;
280         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
281                 if (arg == sym)
282                         return i;
283                 i++;
284         } END_FOR_EACH_PTR(arg);
285
286         return -1;
287 }
288
289 static int format_name_sym_helper(char *buf, int remaining, char *name, struct s
290 {
291         int ret = 0;
292         int arg;
293         char *param_name;
294         int name_len;
295
296         if (!name || !sym || !sym->ident)
297                 goto free;
298         arg = get_param_num_from_sym(sym);
299         if (arg < 0)
300                 goto free;
301         if (param_was_set_var_sym(name, sym))
302                 goto free;
303
304         param_name = sym->ident->name;
305         name_len = strlen(param_name);
306
307         if (name[name_len] == '\0')
308                 ret = snprintf(buf, remaining, "$%d", arg);
309         else if (name[name_len] == '-')
310                 ret = snprintf(buf, remaining, "$%d%s", arg, name + name_len);
311         else
312                 goto free;
313
314         remaining -= ret;
315         if (remaining <= 0)
316                 ret = 0;
317
318 free:
319         free_string(name);
320
321         return ret;
322
323 }
```



```
325 static int format_variable_helper(char *buf, int remaining, struct expression *e
326 {
327         char *name;
328         struct symbol *sym;
329
330         name = expr_to_var_sym(expr, &sym);
331         if (param_was_set_var_sym(name, sym))
332                 return 0;
333         return format_name_sym_helper(buf, remaining, name, sym);
334 }
335
336 static int format_call_to_param_mapping(char *buf, int remaining, struct express
337 {
338         char *name;
339         struct symbol *sym;
340
341         name = map_call_to_param_name_sym(expr, &sym);
342         if (param_was_set_var_sym(name, sym))
343                 return 0;
344         return format_name_sym_helper(buf, remaining, name, sym);
345 }
346
347 static int format_expr_helper(char *buf, int remaining, struct expression *expr)
348 {
349         sval_t sval;
350         int ret;
351         char *cur;
352
353         if (!expr)
354                 return 0;
355
356         cur = buf;
357
358         if (expr->type == EXPR_BINOP) {
359                 ret = format_expr_helper(cur, remaining, expr->left);
360                 if (ret == 0)
361                         return 0;
362                 remaining -= ret;
363                 if (remaining <= 0)
364                         return 0;
365                 cur += ret;
366
367                 ret = snprintf(cur, remaining, " %s ", show_special(expr->op));
368                 remaining -= ret;
369                 if (remaining <= 0)
370                         return 0;
371                 cur += ret;
372
373                 ret = format_expr_helper(cur, remaining, expr->right);
374                 if (ret == 0)
375                         return 0;
376                 remaining -= ret;
377                 if (remaining <= 0)
378                         return 0;
379                 cur += ret;
380                 return cur - buf;
381         }
382
383         if (get_implied_value(expr, &sval)) {
384                 ret = snprintf(cur, remaining, "%s", sval_to_str(sval));
385                 remaining -= ret;
386                 if (remaining <= 0)
387                         return 0;
388                 return ret;
389         }
```

```
391         if (expr->type == EXPR_CALL)
392                 return format_call_to_param_mapping(cur, remaining, expr);
393
394         return format_variable_helper(cur, remaining, expr);
395 }
396
397 static char *format_expr(struct expression *expr)
398 {
399         char buf[256] = "";
400         int ret;
401
402         ret = format_expr_helper(buf, sizeof(buf), expr);
403         if (ret == 0)
404                 return NULL;
405
406         return alloc_sname(buf);
407 }
408
409 char *get_value_in_terms_of_parameter_math(struct expression *expr)
410 {
411         struct expression *tmp;
412         char buf[256] = "";
413         sval_t dummy;
414         int ret;
415
416         tmp = get_assigned_expr(expr);
417         if (tmp)
418                 expr = tmp;
419         if (param_was_set(expr))
420                 return NULL;
421
422         if (get_implied_value(expr, &dummy))
423                 return NULL;
424
425         ret = format_expr_helper(buf, sizeof(buf), expr);
426         if (ret == 0)
427                 return NULL;
428
429         return alloc_sname(buf);
430 }
431
432 char *get_value_in_terms_of_parameter_math_var_sym(const char *name, struct symb
433 {
434         struct expression *tmp, *expr;
435         char buf[256] = "";
436         int ret;
437         int cnt = 0;
438
439         expr = get_assigned_expr_name_sym(name, sym);
440         if (!expr)
441                 return NULL;
442         while ((tmp = get_assigned_expr(expr))) {
443                 expr = strip_expr(tmp);
444                 if (++cnt > 3)
445                         break;
446         }
447
448         ret = format_expr_helper(buf, sizeof(buf), expr);
449         if (ret == 0)
450                 return NULL;
451
452         return alloc_sname(buf);
453
454 }
455
456 static void match_alloc(const char *fn, struct expression *expr, void *_size_arg
```



```
457 {
458         int size_arg = PTR_INT(_size_arg);
459         struct expression *right;
460         struct expression *size_expr;
461         char *sname;
462
463         right = strip_expr(expr->right);
464         size_expr = get_argument_from_call_expr(right->args, size_arg);
465
466         sname = format_expr(size_expr);
467         if (!sname)
468                 return;
469         set_state_expr(my_id, expr->left, alloc_state_sname(sname));
470 }
471
472 static char *swap_format(struct expression *call, char *format)
473 {
474         char buf[256];
475         sval_t sval;
476         long param;
477         struct expression *arg;
478         char *p;
479         char *out;
480         int ret;
481
482         if (format[0] == '$' && format[2] == '\0') {
483                 param = strtol(format + 1, NULL, 10);
484                 arg = get_argument_from_call_expr(call->args, param);
485                 if (!arg)
486                         return NULL;
487                 return format_expr(arg);
488         }
489
490         buf[0] = '\0';
491         p = format;
492         out = buf;
493         while (*p) {
494                 if (*p == '$') {
495                         p++;
496                         param = strtol(p, &p, 10);
497                         arg = get_argument_from_call_expr(call->args, param);
498                         if (!arg)
499                                 return NULL;
500                         param = get_arg_number(arg);
501                         if (param >= 0) {
502                                 ret = snprintf(out, buf + sizeof(buf) - out, "$%
503                                 out += ret;
504                                 if (out >= buf + sizeof(buf))
505                                         return NULL;
506                         } else if (get_implied_value(arg, &sval)) {
507                                 ret = snprintf(out, buf + sizeof(buf) - out, "%s
508                                 out += ret;
509                                 if (out >= buf + sizeof(buf))
510                                         return NULL;
511                         } else {
512                                 return NULL;
513                         }
514                 }
515                 *out = *p;
516                 p++;
517                 out++;
518         }
519         if (buf[0] == '\0')
520                 return NULL;
521         *out = '\0';
522         return alloc_sname(buf);
```

```
}

static char *buf_size_recipe;
static int db_buf_size_callback(void *unused, int argc, char **argv, char **azCo
{
        if (argc != 1)
                return 0;

        if (!buf_size_recipe)
                buf_size_recipe = alloc_sname(argv[0]);
        else if (strcmp(buf_size_recipe, argv[0]) != 0)
                buf_size_recipe = alloc_sname("invalid");
        return 0;
}

static char *get_allocation_recipe_from_call(struct expression *expr)
{
        struct symbol *sym;
        static char sql_filter[1024];
        int i;

        if (is_fake_call(expr))
                return NULL;
        expr = strip_expr(expr);
        if (expr->fn->type != EXPR_SYMBOL)
                return NULL;
        sym = expr->fn->symbol;
        if (!sym)
                return NULL;

        for (i = 0; i < ARRAY_SIZE(alloc_functions); i++) {
                if (strcmp(sym->ident->name, alloc_functions[i].func) == 0) {
                        char buf[32];

                        snprintf(buf, sizeof(buf), "$%d", alloc_functions[i].par
                        buf_size_recipe = alloc_sname(buf);
                        return swap_format(expr, buf_size_recipe);
                }
        }

        if (sym->ctype.modifiers & MOD_STATIC) {
                snprintf(sql_filter, 1024, "file = '%s' and function = '%s';",
                         get_filename(), sym->ident->name);
        } else {
                snprintf(sql_filter, 1024, "function = '%s' and static = 0;",
                                sym->ident->name);
        }

        buf_size_recipe = NULL;
        run_sql(db_buf_size_callback, NULL,
                "select value from return_states where type=%d and %s",
                BUF_SIZE, sql_filter);
        if (!buf_size_recipe || strcmp(buf_size_recipe, "invalid") == 0)
                return NULL;
        return swap_format(expr, buf_size_recipe);
}

static void match_call_assignment(struct expression *expr)
{
        char *sname;

        sname = get_allocation_recipe_from_call(expr->right);
        if (!sname)
                return;
        set_state_expr(my_id, expr->left, alloc_state_sname(sname));
}
```



```
static void match_returns_call(int return_id, char *return_ranges, struct expres
{
        char *sname;

        sname = get_allocation_recipe_from_call(call);
        if (option_debug)
                sm_msg("sname = %s", sname);
        if (!sname)
                return;

        sql_insert_return_states(return_id, return_ranges, BUF_SIZE, -1, "",
                        sname);
}

static void print_returned_allocations(int return_id, char *return_ranges, struc
{
        struct expression *tmp;
        struct smatch_state *state;
        struct symbol *sym;
        char *name;
        int cnt = 0;

        expr = strip_expr(expr);
        while ((tmp = get_assigned_expr(expr))) {
                if (cnt++ > 5)  /* assignments to self cause infinite loops */
                        break;
                expr = strip_expr(tmp);
        }
        if (!expr)
                return;

        if (expr->type == EXPR_CALL) {
                match_returns_call(return_id, return_ranges, expr);
                return;
        }

        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                goto free;

        state = get_state(my_id, name, sym);
        if (!state || !state->data)
                goto free;

        sql_insert_return_states(return_id, return_ranges, BUF_SIZE, -1, "",
                        state->name);
free:
        free_string(name);
}

void register_parse_call_math(int id)
{
        int i;

        my_id = id;

        for (i = 0; i < ARRAY_SIZE(alloc_functions); i++)
                add_function_assign_hook(alloc_functions[i].func, &match_alloc,
                                         INT_PTR(alloc_functions[i].param));
        add_hook(&match_call_assignment, CALL_ASSIGNMENT_HOOK);
        add_split_return_callback(print_returned_allocations);
}
```

```
**********************************************************
    1900 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_passes_array_size.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_extra.h"

static int find_param_eq(struct expression *expr, int size)
{
        struct expression *arg;
        sval_t val;
        int i;

        i = -1;
        FOR_EACH_PTR(expr->args, arg) {
                i++;
                if (!get_implied_value(arg, &val))
                        continue;
                if (val.value == size)
                        return i;
        } END_FOR_EACH_PTR(arg);

        return -1;
}

static void match_call(struct expression *expr)
{
        struct expression *arg;
        struct symbol *type;
        int size, bytes;
        int i, nr;
        char buf[16];


        i = -1;
        FOR_EACH_PTR(expr->args, arg) {
                i++;
                type = get_type(arg);
                if (!type || (type->type != SYM_PTR && type->type != SYM_ARRAY))
                        continue;
                size = get_array_size(arg);
                if (size > 0) {
                        nr = find_param_eq(expr, size);
                        if (nr >= 0) {
                                snprintf(buf, sizeof(buf), "%d", nr);
                                sql_insert_caller_info(expr, ARRAYSIZE_ARG, i, b
                                continue;
```



```
                        }
                }
                bytes = get_array_size_bytes(arg);
                if (bytes > 0) {
                        nr = find_param_eq(expr, bytes);
                        if (nr >= 0) {
                                snprintf(buf, sizeof(buf), "%d", nr);
                                sql_insert_caller_info(expr, SIZEOF_ARG, i, buf,
                                continue;
                        }
                }
        } END_FOR_EACH_PTR(arg);
}

void register_passes_array_size(int id)
{
        add_hook(&match_call, FUNCTION_CALL_HOOK);
}
```

```
**********************************************************
    5068 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_project.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2010 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This file is only for very generic stuff, that is reusable
 * between projects.  If you need something special create a
 * check_your_project.c.
 *
 */

#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_function_hashtable.h"

static DEFINE_HASHTABLE_INSERT(insert_func, char, int);
static DEFINE_HASHTABLE_SEARCH(search_func, char, int);
static struct hashtable *skipped_funcs;
static struct hashtable *silenced_funcs;
static struct hashtable *no_inline_funcs;

int is_skipped_function(void)
{
        char *func;

        func = get_function();
        if (!func)
                return 0;
        if (search_func(skipped_funcs, func))
                return 1;
        return 0;
}

/*
 * A silenced function will still be processed and potentially appear in info
 * output, but not regular checks.
 */
int is_silenced_function(void)
{
        char *func;

        if (is_skipped_function())
                return 1;

        func = get_function();
        if (!func)
                return 0;
```



```
        if (search_func(silenced_funcs, func))
                return 1;
        return 0;
}

int is_no_inline_function(const char *function)
{
        if (search_func(no_inline_funcs, (char *)function))
                return 1;
        return 0;
}

static void register_no_return_funcs(void)
{
        struct token *token;
        const char *func;
        char name[256];

        snprintf(name, 256, "%s.no_return_funcs", option_project_str);

        token = get_tokens_file(name);
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                func = show_ident(token->ident);
                add_function_hook(func, &__match_nullify_path_hook, NULL);
                token = token->next;
        }
        clear_token_alloc();
}

static void register_ignored_macros(void)
{
        struct token *token;
        char *macro;
        char name[256];

        if (option_project == PROJ_NONE)
                strcpy(name, "ignored_macros");
        else
                snprintf(name, 256, "%s.ignored_macros", option_project_str);

        token = get_tokens_file(name);
        if (!token)
                return;
        if (token_type(token) != TOKEN_STREAMBEGIN)
                return;
        token = token->next;
        while (token_type(token) != TOKEN_STREAMEND) {
                if (token_type(token) != TOKEN_IDENT)
                        return;
                macro = alloc_string(show_ident(token->ident));
                add_ptr_list(&__ignored_macros, macro);
                token = token->next;
        }
        clear_token_alloc();
}

static void register_skipped_functions(void)
{
        struct token *token;
```

```
127         char *func;
128         char name[256];

130         skipped_funcs = create_function_hashtable(500);

132         if (option_project == PROJ_NONE)
133                 return;

135         snprintf(name, 256, "%s.skipped_functions", option_project_str);

137         token = get_tokens_file(name);
138         if (!token)
139                 return;
140         if (token_type(token) != TOKEN_STREAMBEGIN)
141                 return;
142         token = token->next;
143         while (token_type(token) != TOKEN_STREAMEND) {
144                 if (token_type(token) != TOKEN_IDENT)
145                         return;
146                 func = alloc_string(show_ident(token->ident));
147                 insert_func(skipped_funcs, func, INT_PTR(1));
148                 token = token->next;
149         }
150         clear_token_alloc();
151 }

153 static void register_silenced_functions(void)
154 {
155         struct token *token;
156         char *func;
157         char name[256];

159         silenced_funcs = create_function_hashtable(500);

161         if (option_project == PROJ_NONE)
162                 return;

164         snprintf(name, 256, "%s.silenced_functions", option_project_str);

166         token = get_tokens_file(name);
167         if (!token)
168                 return;
169         if (token_type(token) != TOKEN_STREAMBEGIN)
170                 return;
171         token = token->next;
172         while (token_type(token) != TOKEN_STREAMEND) {
173                 if (token_type(token) != TOKEN_IDENT)
174                         return;
175                 func = alloc_string(show_ident(token->ident));
176                 insert_func(silenced_funcs, func, INT_PTR(1));
177                 token = token->next;
178         }
179         clear_token_alloc();
180 }

182 static void register_no_inline_functions(void)
183 {
184         struct token *token;
185         char *func;
186         char name[256];

188         no_inline_funcs = create_function_hashtable(500);

190         if (option_project == PROJ_NONE)
191                 return;
```



```
193         snprintf(name, 256, "%s.no_inline_functions", option_project_str);

195         token = get_tokens_file(name);
196         if (!token)
197                 return;
198         if (token_type(token) != TOKEN_STREAMBEGIN)
199                 return;
200         token = token->next;
201         while (token_type(token) != TOKEN_STREAMEND) {
202                 if (token_type(token) != TOKEN_IDENT)
203                         return;
204                 func = alloc_string(show_ident(token->ident));
205                 insert_func(no_inline_funcs, func, INT_PTR(1));
206                 token = token->next;
207         }
208         clear_token_alloc();
209 }

211 void register_project(int id)
212 {
213         register_no_return_funcs();
214         register_ignored_macros();
215         register_skipped_functions();
216         register_silenced_functions();
217         register_no_inline_functions();
218 }
```

```
**********************************************************
   45585 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_ranges.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "parse.h"
#include "smatch.h"
#include "smatch_extra.h"
#include "smatch_slist.h"

ALLOCATOR(data_info, "smatch extra data");
ALLOCATOR(data_range, "data range");
__DO_ALLOCATOR(struct data_range, sizeof(struct data_range), __alignof__(struct
                        "permanent ranges", perm_data_range);
__DECLARE_ALLOCATOR(struct ptr_list, rl_ptrlist);

char *show_rl(struct range_list *list)
{
        struct data_range *tmp;
        char full[512];
        int i = 0;

        full[0] = '\0';
        full[sizeof(full) - 1] = '\0';
        FOR_EACH_PTR(list, tmp) {
                if (i++)
                        strncat(full, ",", 254 - strlen(full));
                if (sval_cmp(tmp->min, tmp->max) == 0) {
                        strncat(full, sval_to_str(tmp->min), 254 - strlen(full));
                        continue;
                }
                strncat(full, sval_to_str(tmp->min), 254 - strlen(full));
                strncat(full, "-", 254 - strlen(full));
                strncat(full, sval_to_str(tmp->max), 254 - strlen(full));
        } END_FOR_EACH_PTR(tmp);
        if (strlen(full) == sizeof(full) - 1)
                full[sizeof(full) - 2] = '+';
        return alloc_sname(full);
}

void free_all_rl(void)
{
        clear_rl_ptrlist_alloc();
}

static int sval_too_big(struct symbol *type, sval_t sval)
{
        if (type_bits(type) >= 32 &&
```



```
            type_bits(sval.type) <= type_bits(type))
                return 0;
        if (sval.uvalue <= ((1ULL << type_bits(type)) - 1))
                return 0;
        if (type_signed(sval.type)) {
                if (type_unsigned(type)) {
                        unsigned long long neg = ~sval.uvalue;
                        if (neg <= sval_type_max(type).uvalue)
                                return 0;
                }
                if (sval.value < sval_type_min(type).value)
                        return 1;
                if (sval.value > sval_type_max(type).value)
                        return 1;
                return 0;
        }
        if (sval.uvalue > sval_type_max(type).uvalue)
                return 1;
        return 0;
}

static void add_range_t(struct symbol *type, struct range_list **rl, sval_t min,
{
        /* If we're just adding a number, cast it and add it */
        if (sval_cmp(min, max) == 0) {
                add_range(rl, sval_cast(type, min), sval_cast(type, max));
                return;
        }

        /* If the range is within the type range then add it */
        if (sval_fits(type, min) && sval_fits(type, max)) {
                add_range(rl, sval_cast(type, min), sval_cast(type, max));
                return;
        }

        /*
         * If the range we are adding has more bits than the range type then
         * add the whole range type.  Eg:
         * 0x8000000000000000 - 0xf000000000000000 -> cast to int
         * This isn't totally the right thing to do.  We could be more granular.
         */
        if (sval_too_big(type, min) || sval_too_big(type, max)) {
                add_range(rl, sval_type_min(type), sval_type_max(type));
                return;
        }

        /* Cast negative values to high positive values */
        if (sval_is_negative(min) && type_unsigned(type)) {
                if (sval_is_positive(max)) {
                        if (sval_too_high(type, max)) {
                                add_range(rl, sval_type_min(type), sval_type_max
                                return;
                        }
                        add_range(rl, sval_type_val(type, 0), sval_cast(type, ma
                        max = sval_type_max(type);
                } else {
                        max = sval_cast(type, max);
                }
                min = sval_cast(type, min);
                add_range(rl, min, max);
        }

        /* Cast high positive numbers to negative */
        if (sval_unsigned(max) && sval_is_negative(sval_cast(type, max))) {
                if (!sval_is_negative(sval_cast(type, min))) {
                        add_range(rl, sval_cast(type, min), sval_type_max(type))
```

```
127                         min = sval_type_min(type);
128                 } else {
129                         min = sval_cast(type, min);
130                 }
131                 max = sval_cast(type, max);
132                 add_range(rl, min, max);
133         }

135         add_range(rl, sval_cast(type, min), sval_cast(type, max));
136         return;
137 }

139 static int str_to_comparison_arg_helper(const char *str,
140                 struct expression *call, int *comparison,
141                 struct expression **arg, char **endp)
142 {
143         int param;
144         char *c = (char *)str;

146         if (*c != '[')
147                 return 0;
148         c++;

150         if (*c == '<') {
151                 c++;
152                 if (*c == '=') {
153                         *comparison = SPECIAL_LTE;
154                         c++;
155                 } else {
156                         *comparison = '<';
157                 }
158         } else if (*c == '=') {
159                 c++;
160                 c++;
161                 *comparison = SPECIAL_EQUAL;
162         } else if (*c == '>') {
163                 c++;
164                 if (*c == '=') {
165                         *comparison = SPECIAL_GTE;
166                         c++;
167                 } else {
168                         *comparison = '>';
169                 }
170         } else if (*c == '!') {
171                 c++;
172                 c++;
173                 *comparison = SPECIAL_NOTEQUAL;
174         } else {
175                 return 0;
176         }

178         if (*c != '$')
179                 return 0;
180         c++;

182         param = strtoll(c, &c, 10);
183         if (*c == ']')
184                 c++; /* skip the ']' character */
185         if (endp)
186                 *endp = (char *)c;

188         if (!call)
189                 return 0;
190         *arg = get_argument_from_call_expr(call->args, param);
191         if (!*arg)
192                 return 0;
```



```
193         if (*c == '-' && *(c + 1) == '>') {
194                 char buf[256];
195                 int n;

197                 n = snprintf(buf, sizeof(buf), "$%s", c);
198                 if (n >= sizeof(buf))
199                         return 0;
200                 if (buf[n - 1] == ']')
201                         buf[n - 1] = '\0';
202                 *arg = gen_expression_from_key(*arg, buf);
203                 while (*c && *c != ']')
204                         c++;
205         }
206         return 1;
207 }

209 int str_to_comparison_arg(const char *str, struct expression *call, int *compari
210 {
211         while (1) {
212                 if (!*str)
213                         return 0;
214                 if (*str == '[')
215                         break;
216                 str++;
217         }
218         return str_to_comparison_arg_helper(str, call, comparison, arg, NULL);
219 }

221 static int get_val_from_key(int use_max, struct symbol *type, char *c, struct ex
222 {
223         struct expression *arg;
224         int comparison;
225         sval_t ret, tmp;

227         if (use_max)
228                 ret = sval_type_max(type);
229         else
230                 ret = sval_type_min(type);

232         if (!str_to_comparison_arg_helper(c, call, &comparison, &arg, endp)) {
233                 *sval = ret;
234                 return 0;
235         }

237         if (use_max && get_implied_max(arg, &tmp)) {
238                 ret = tmp;
239                 if (comparison == '<') {
240                         tmp.value = 1;
241                         ret = sval_binop(ret, '-', tmp);
242                 }
243         }
244         if (!use_max && get_implied_min(arg, &tmp)) {
245                 ret = tmp;
246                 if (comparison == '>') {
247                         tmp.value = 1;
248                         ret = sval_binop(ret, '+', tmp);
249                 }
250         }

252         *sval = ret;
253         return 1;
254 }

256 static sval_t add_one(sval_t sval)
257 {
258         sval.value++;
```

```
259         return sval;
260 }

262 static sval_t sub_one(sval_t sval)
263 {
264         sval.value--;
265         return sval;
266 }

268 void filter_by_comparison(struct range_list **rl, int comparison, struct range_l
269 {
270         struct range_list *left_orig = *rl;
271         struct range_list *right_orig = right;
272         struct range_list *ret_rl = *rl;
273         struct symbol *cast_type;
274         sval_t min, max;

276         cast_type = rl_type(left_orig);
277         if (sval_type_max(rl_type(left_orig)).uvalue < sval_type_max(rl_type(rig
278                 cast_type = rl_type(right_orig);
279         if (sval_type_max(cast_type).uvalue < INT_MAX)
280                 cast_type = &int_ctype;

282         min = sval_type_min(cast_type);
283         max = sval_type_max(cast_type);
284         left_orig = cast_rl(cast_type, left_orig);
285         right_orig = cast_rl(cast_type, right_orig);

287         switch (comparison) {
288         case '<':
289         case SPECIAL_UNSIGNED_LT:
290                 ret_rl = remove_range(left_orig, rl_max(right_orig), max);
291                 break;
292         case SPECIAL_LTE:
293         case SPECIAL_UNSIGNED_LTE:
294                 if (!sval_is_max(rl_max(right_orig)))
295                         ret_rl = remove_range(left_orig, add_one(rl_max(right_or
296                 break;
297         case SPECIAL_EQUAL:
298                 ret_rl = rl_intersection(left_orig, right_orig);
299                 break;
300         case SPECIAL_GTE:
301         case SPECIAL_UNSIGNED_GTE:
302                 if (!sval_is_min(rl_min(right_orig)))
303                         ret_rl = remove_range(left_orig, min, sub_one(rl_min(rig
304                 break;
305         case '>':
306         case SPECIAL_UNSIGNED_GT:
307                 ret_rl = remove_range(left_orig, min, rl_min(right_orig));
308                 break;
309         case SPECIAL_NOTEQUAL:
310                 if (sval_cmp(rl_min(right_orig), rl_max(right_orig)) == 0)
311                         ret_rl = remove_range(left_orig, rl_min(right_orig), rl_
312                 break;
313         default:
314                 sm_perror("unhandled comparison %s", show_special(comparison));
315                 return;
316         }

318         *rl = cast_rl(rl_type(*rl), ret_rl);
319 }

321 static struct range_list *filter_by_comparison_call(char *c, struct expression *
322 {
323         struct symbol *type;
324         struct expression *arg;
```



```
325         struct range_list *casted_start, *right_orig;
326         int comparison;

328         if (!str_to_comparison_arg_helper(c, call, &comparison, &arg, endp))
329                 return start_rl;

331         if (!get_implied_rl(arg, &right_orig))
332                 return start_rl;

334         type = &int_ctype;
335         if (type_positive_bits(rl_type(start_rl)) > type_positive_bits(type))
336                 type = rl_type(start_rl);
337         if (type_positive_bits(rl_type(right_orig)) > type_positive_bits(type))
338                 type = rl_type(right_orig);

340         casted_start = cast_rl(type, start_rl);
341         right_orig = cast_rl(type, right_orig);

343         filter_by_comparison(&casted_start, comparison, right_orig);
344         return cast_rl(rl_type(start_rl), casted_start);
345 }

347 static sval_t parse_val(int use_max, struct expression *call, struct symbol *typ
348 {
349         char *start = c;
350         sval_t ret;

352         if (!strncmp(start, "max", 3)) {
353                 ret = sval_type_max(type);
354                 c += 3;
355         } else if (!strncmp(start, "u64max", 6)) {
356                 ret = sval_type_val(type, ULLONG_MAX);
357                 c += 6;
358         } else if (!strncmp(start, "s64max", 6)) {
359                 ret = sval_type_val(type, LLONG_MAX);
360                 c += 6;
361         } else if (!strncmp(start, "u32max", 6)) {
362                 ret = sval_type_val(type, UINT_MAX);
363                 c += 6;
364         } else if (!strncmp(start, "s32max", 6)) {
365                 ret = sval_type_val(type, INT_MAX);
366                 c += 6;
367         } else if (!strncmp(start, "u16max", 6)) {
368                 ret = sval_type_val(type, USHRT_MAX);
369                 c += 6;
370         } else if (!strncmp(start, "s16max", 6)) {
371                 ret = sval_type_val(type, SHRT_MAX);
372                 c += 6;
373         } else if (!strncmp(start, "min", 3)) {
374                 ret = sval_type_min(type);
375                 c += 3;
376         } else if (!strncmp(start, "s64min", 6)) {
377                 ret = sval_type_val(type, LLONG_MIN);
378                 c += 6;
379         } else if (!strncmp(start, "s32min", 6)) {
380                 ret = sval_type_val(type, INT_MIN);
381                 c += 6;
382         } else if (!strncmp(start, "s16min", 6)) {
383                 ret = sval_type_val(type, SHRT_MIN);
384                 c += 6;
385         } else if (!strncmp(start, "long_min", 8)) {
386                 ret = sval_type_val(type, LONG_MIN);
387                 c += 8;
388         } else if (!strncmp(start, "long_max", 8)) {
389                 ret = sval_type_val(type, LONG_MAX);
390                 c += 8;
```

```
391         } else if (!strncmp(start, "ulong_max", 9)) {
392                 ret = sval_type_val(type, ULONG_MAX);
393                 c += 9;
394         } else if (!strncmp(start, "ptr_max", 7)) {
395                 ret = sval_type_val(type, valid_ptr_max);
396                 c += 7;
397         } else if (start[0] == '[') {
398                 /* this parses [==p0] comparisons */
399                 get_val_from_key(1, type, start, call, &c, &ret);
400         } else if (type_positive_bits(type) == 64) {
401                 ret = sval_type_val(type, strtoull(start, &c, 0));
402         } else {
403                 ret = sval_type_val(type, strtoll(start, &c, 0));
404         }
405         *endp = c;
406         return ret;
407 }

409 static char *jump_to_call_math(char *value)
410 {
411         char *c = value;

413         while (*c && *c != '[')
414                 c++;

416         if (!*c)
417                 return NULL;
418         c++;
419         if (*c == '<' || *c == '=' || *c == '>' || *c == '!')
420                 return NULL;

422         return c;
423 }

425 static void str_to_rl_helper(struct expression *call, struct symbol *type, char
426 {
427         struct range_list *rl_tmp = NULL;
428         sval_t min, max;
429         char *c;

431         min = sval_type_min(type);
432         max = sval_type_max(type);
433         c = str;
434         while (*c != '\0' && *c != '[') {
435                 if (*c == '+') {
436                         if (sval_cmp(min, sval_type_min(type)) != 0)
437                                 min = max;
438                         max = sval_type_max(type);
439                         add_range_t(type, &rl_tmp, min, max);
440                         break;
441                 }
442                 if (*c == '(')
443                         c++;
444                 min = parse_val(0, call, type, c, &c);
445                 if (!sval_fits(type, min))
446                         min = sval_type_min(type);
447                 max = min;
448                 if (*c == ')')
449                         c++;
450                 if (*c == '\0' || *c == '[') {
451                         add_range_t(type, &rl_tmp, min, min);
452                         break;
453                 }
454                 if (*c == ',') {
455                         add_range_t(type, &rl_tmp, min, min);
456                         c++;
```



```
457                         continue;
458                 }
459                 if (*c == '+') {
460                         min = sval_type_max(type);
461                         c++;
462                 }
463                 if (*c != '-') {
464                         sm_msg("debug XXX: trouble parsing %s c = %s", str, c);
465                         break;
466                 }
467                 c++;
468                 if (*c == '('
469                         c++;
470                 max = parse_val(1, call, type, c, &c);
471                 if (!sval_fits(type, max))
472                         max = sval_type_max(type);
473                 if (*c == '+') {
474                         max = sval_type_max(type);
475                         c++;
476                 }
477                 add_range_t(type, &rl_tmp, min, max);
478                 if (*c == ')')
479                         c++;
480                 if (*c == ',')
481                         c++;
482         }

484         *rl = rl_tmp;
485         *endp = c;
486 }

488 static void str_to_dinfo(struct expression *call, struct symbol *type, char *val
489 {
490         struct range_list *math_rl;
491         char *call_math;
492         char *c;
493         struct range_list *rl = NULL;

495         if (!type)
496                 type = &llong_ctype;

498         if (strcmp(value, "empty") == 0)
499                 return;

501         if (strncmp(value, "[==$", 4) == 0) {
502                 struct expression *arg;
503                 int comparison;

505                 if (!str_to_comparison_arg(value, call, &comparison, &arg))
506                         return;
507                 if (!get_implied_rl(arg, &rl))
508                         return;
509                 goto cast;
510         }

512         str_to_rl_helper(call, type, value, &c, &rl);
513         if (*c == '\0')
514                 goto cast;

516         call_math = jump_to_call_math(value);
517         if (call_math && parse_call_math_rl(call, call_math, &math_rl)) {
518                 rl = rl_intersection(rl, math_rl);
519                 goto cast;
520         }

522         /*
```

```
523          * For now if we already tried to handle the call math and couldn't
524          * figure it out then bail.
525          */
526         if (jump_to_call_math(c) == c + 1)
527                 goto cast;

529         rl = filter_by_comparison_call(c, call, &c, rl);

531 cast:
532         rl = cast_rl(type, rl);
533         dinfo->value_ranges = rl;
534 }

536 void str_to_rl(struct symbol *type, char *value, struct range_list **rl)
537 {
538         struct data_info dinfo = {};

540         str_to_dinfo(NULL, type, value, &dinfo);
541         *rl = dinfo.value_ranges;
542 }

544 void call_results_to_rl(struct expression *expr, struct symbol *type, char *valu
545 {
546         struct data_info dinfo = {};

548         str_to_dinfo(strip_expr(expr), type, value, &dinfo);
549         *rl = dinfo.value_ranges;
550 }

552 int is_whole_rl(struct range_list *rl)
553 {
554         struct data_range *drange;

556         if (ptr_list_empty(rl))
557                 return 0;
558         drange = first_ptr_list((struct ptr_list *)rl);
559         if (sval_is_min(drange->min) && sval_is_max(drange->max))
560                 return 1;
561         return 0;
562 }

564 int is_whole_rl_non_zero(struct range_list *rl)
565 {
566         struct data_range *drange;

568         if (ptr_list_empty(rl))
569                 return 0;
570         drange = first_ptr_list((struct ptr_list *)rl);
571         if (sval_unsigned(drange->min) &&
572             drange->min.value == 1 &&
573             sval_is_max(drange->max))
574                 return 1;
575         if (!sval_is_min(drange->min) || drange->max.value != -1)
576                 return 0;
577         drange = last_ptr_list((struct ptr_list *)rl);
578         if (drange->min.value != 1 || !sval_is_max(drange->max))
579                 return 0;
580         return 1;
581 }

583 sval_t rl_min(struct range_list *rl)
584 {
585         struct data_range *drange;
586         sval_t ret;

588         ret.type = &llong_ctype;
```



```
589         ret.value = LLONG_MIN;
590         if (ptr_list_empty(rl))
591                 return ret;
592         drange = first_ptr_list((struct ptr_list *)rl);
593         return drange->min;
594 }

596 sval_t rl_max(struct range_list *rl)
597 {
598         struct data_range *drange;
599         sval_t ret;

601         ret.type = &llong_ctype;
602         ret.value = LLONG_MAX;
603         if (ptr_list_empty(rl))
604                 return ret;
605         drange = last_ptr_list((struct ptr_list *)rl);
606         return drange->max;
607 }

609 int rl_to_sval(struct range_list *rl, sval_t *sval)
610 {
611         sval_t min, max;

613         if (!rl)
614                 return 0;

616         min = rl_min(rl);
617         max = rl_max(rl);
618         if (sval_cmp(min, max) != 0)
619                 return 0;
620         *sval = min;
621         return 1;
622 }

624 struct symbol *rl_type(struct range_list *rl)
625 {
626         if (!rl)
627                 return NULL;
628         return rl_min(rl).type;
629 }

631 static struct data_range *alloc_range_helper_sval(sval_t min, sval_t max, int pe
632 {
633         struct data_range *ret;

635         if (perm)
636                 ret = __alloc_perm_data_range(0);
637         else
638                 ret = __alloc_data_range(0);
639         ret->min = min;
640         ret->max = max;
641         return ret;
642 }

644 struct data_range *alloc_range(sval_t min, sval_t max)
645 {
646         return alloc_range_helper_sval(min, max, 0);
647 }

649 struct data_range *alloc_range_perm(sval_t min, sval_t max)
650 {
651         return alloc_range_helper_sval(min, max, 1);
652 }

654 struct range_list *alloc_rl(sval_t min, sval_t max)
```

```
655 {
656         struct range_list *rl = NULL;

658         if (sval_cmp(min, max) > 0)
659                 return alloc_whole_rl(min.type);

661         add_range(&rl, min, max);
662         return rl;
663 }

665 struct range_list *alloc_whole_rl(struct symbol *type)
666 {
667         if (!type || type_positive_bits(type) < 0)
668                 type = &llong_ctype;
669         if (type->type == SYM_ARRAY)
670                 type = &ptr_ctype;

672         return alloc_rl(sval_type_min(type), sval_type_max(type));
673 }

675 extern int rl_ptrlist_hack;
676 void add_range(struct range_list **list, sval_t min, sval_t max)
677 {
678         struct data_range *tmp;
679         struct data_range *new = NULL;
680         int check_next = 0;

682         /*
683          * There is at least on valid reason why the types might be confusing
684          * and that's when you have a void pointer and on some paths you treat
685          * it as a u8 pointer and on other paths you treat it as a u16 pointer.
686          * This case is hard to deal with.
687          *
688          * There are other cases where we probably should be more specific about
689          * the types than we are.  For example, we end up merging a lot of ulong
690          * with pointers and I have not figured out why we do that.
691          *
692          * But this hack works for both cases, I think.  We cast it to pointers
693          * or we use the bigger size.
694          *
695          */
696         if (*list && rl_type(*list) != min.type) {
697                 if (rl_type(*list)->type == SYM_PTR) {
698                         min = sval_cast(rl_type(*list), min);
699                         max = sval_cast(rl_type(*list), max);
700                 } else if (min.type->type == SYM_PTR) {
701                         *list = cast_rl(min.type, *list);
702                 } else if (type_bits(rl_type(*list)) >= type_bits(min.type)) {
703                         min = sval_cast(rl_type(*list), min);
704                         max = sval_cast(rl_type(*list), max);
705                 } else {
706                         *list = cast_rl(min.type, *list);
707                 }
708         }

710         if (sval_cmp(min, max) > 0) {
711                 min = sval_type_min(min.type);
712                 max = sval_type_max(min.type);
713         }

715         /*
716          * FIXME:  This has a problem merging a range_list like: min-0,3-max
717          * with a range like 1-2.  You end up with min-2,3-max instead of
718          * just min-max.
719          */
720         FOR_EACH_PTR(*list, tmp) {
```



```
721                 if (check_next) {
722                         /* Sometimes we overlap with more than one range
723                            so we have to delete or modify the next range. */
724                         if (!sval_is_max(max) && max.value + 1 == tmp->min.value
725                                 /* join 2 ranges here */
726                                 new->max = tmp->max;
727                                 DELETE_CURRENT_PTR(tmp);
728                                 return;
729                         }

731                         /* Doesn't overlap with the next one. */
732                         if (sval_cmp(max, tmp->min) < 0)
733                                 return;

735                         if (sval_cmp(max, tmp->max) <= 0) {
736                                 /* Partially overlaps the next one. */
737                                 new->max = tmp->max;
738                                 DELETE_CURRENT_PTR(tmp);
739                                 return;
740                         } else {
741                                 /* Completely overlaps the next one. */
742                                 DELETE_CURRENT_PTR(tmp);
743                                 /* there could be more ranges to delete */
744                                 continue;
745                         }
746                 }
747                 if (!sval_is_max(max) && max.value + 1 == tmp->min.value) {
748                         /* join 2 ranges into a big range */
749                         new = alloc_range(min, tmp->max);
750                         REPLACE_CURRENT_PTR(tmp, new);
751                         return;
752                 }
753                 if (sval_cmp(max, tmp->min) < 0) { /* new range entirely below *
754                         new = alloc_range(min, max);
755                         INSERT_CURRENT(new, tmp);
756                         return;
757                 }
758                 if (sval_cmp(min, tmp->min) < 0) { /* new range partially below
759                         if (sval_cmp(max, tmp->max) < 0)
760                                 max = tmp->max;
761                         else
762                                 check_next = 1;
763                         new = alloc_range(min, max);
764                         REPLACE_CURRENT_PTR(tmp, new);
765                         if (!check_next)
766                                 return;
767                         continue;
768                 }
769                 if (sval_cmp(max, tmp->max) <= 0) /* new range already included
770                         return;
771                 if (sval_cmp(min, tmp->max) <= 0) { /* new range partially above
772                         min = tmp->min;
773                         new = alloc_range(min, max);
774                         REPLACE_CURRENT_PTR(tmp, new);
775                         check_next = 1;
776                         continue;
777                 }
778                 if (!sval_is_min(min) && min.value - 1 == tmp->max.value) {
779                         /* join 2 ranges into a big range */
780                         new = alloc_range(tmp->min, max);
781                         REPLACE_CURRENT_PTR(tmp, new);
782                         check_next = 1;
783                         continue;
784                 }
785                 /* the new range is entirely above the existing ranges */
786         } END_FOR_EACH_PTR(tmp);
```

```
787         if (check_next)
788                 return;
789         new = alloc_range(min, max);
790
791         rl_ptrlist_hack = 1;
792         add_ptr_list(list, new);
793         rl_ptrlist_hack = 0;
794 }
795
796 struct range_list *clone_rl(struct range_list *list)
797 {
798         struct data_range *tmp;
799         struct range_list *ret = NULL;
800
801         FOR_EACH_PTR(list, tmp) {
802                 add_ptr_list(&ret, tmp);
803         } END_FOR_EACH_PTR(tmp);
804         return ret;
805 }
806
807 struct range_list *clone_rl_permanent(struct range_list *list)
808 {
809         struct data_range *tmp;
810         struct data_range *new;
811         struct range_list *ret = NULL;
812
813         FOR_EACH_PTR(list, tmp) {
814                 new = alloc_range_perm(tmp->min, tmp->max);
815                 add_ptr_list(&ret, new);
816         } END_FOR_EACH_PTR(tmp);
817         return ret;
818 }
819
820 struct range_list *rl_union(struct range_list *one, struct range_list *two)
821 {
822         struct data_range *tmp;
823         struct range_list *ret = NULL;
824
825         FOR_EACH_PTR(one, tmp) {
826                 add_range(&ret, tmp->min, tmp->max);
827         } END_FOR_EACH_PTR(tmp);
828         FOR_EACH_PTR(two, tmp) {
829                 add_range(&ret, tmp->min, tmp->max);
830         } END_FOR_EACH_PTR(tmp);
831         return ret;
832 }
833
834 struct range_list *remove_range(struct range_list *list, sval_t min, sval_t max)
835 {
836         struct data_range *tmp;
837         struct range_list *ret = NULL;
838
839         if (!list)
840                 return NULL;
841
842         min = sval_cast(rl_type(list), min);
843         max = sval_cast(rl_type(list), max);
844         if (sval_cmp(min, max) > 0) {
845                 sval_t tmp = min;
846                 min = max;
847                 max = tmp;
848         }
849
850         FOR_EACH_PTR(list, tmp) {
851                 if (sval_cmp(tmp->max, min) < 0) {
852                         add_range(&ret, tmp->min, tmp->max);
```



```
853                         continue;
854                 }
855                 if (sval_cmp(tmp->min, max) > 0) {
856                         add_range(&ret, tmp->min, tmp->max);
857                         continue;
858                 }
859                 if (sval_cmp(tmp->min, min) >= 0 && sval_cmp(tmp->max, max) <= 0
860                         continue;
861                 if (sval_cmp(tmp->min, min) >= 0) {
862                         max.value++;
863                         add_range(&ret, max, tmp->max);
864                 } else if (sval_cmp(tmp->max, max) <= 0) {
865                         min.value--;
866                         add_range(&ret, tmp->min, min);
867                 } else {
868                         min.value--;
869                         max.value++;
870                         add_range(&ret, tmp->min, min);
871                         add_range(&ret, max, tmp->max);
872                 }
873         } END_FOR_EACH_PTR(tmp);
874         return ret;
875 }
876
877 int ranges_equiv(struct data_range *one, struct data_range *two)
878 {
879         if (!one && !two)
880                 return 1;
881         if (!one || !two)
882                 return 0;
883         if (sval_cmp(one->min, two->min) != 0)
884                 return 0;
885         if (sval_cmp(one->max, two->max) != 0)
886                 return 0;
887         return 1;
888 }
889
890 int rl_equiv(struct range_list *one, struct range_list *two)
891 {
892         struct data_range *one_range;
893         struct data_range *two_range;
894
895         if (one == two)
896                 return 1;
897
898         PREPARE_PTR_LIST(one, one_range);
899         PREPARE_PTR_LIST(two, two_range);
900         for (;;) {
901                 if (!one_range && !two_range)
902                         return 1;
903                 if (!ranges_equiv(one_range, two_range))
904                         return 0;
905                 NEXT_PTR_LIST(one_range);
906                 NEXT_PTR_LIST(two_range);
907         }
908         FINISH_PTR_LIST(two_range);
909         FINISH_PTR_LIST(one_range);
910
911         return 1;
912 }
913
914 int true_comparison_range(struct data_range *left, int comparison, struct data_r
915 {
916         switch (comparison) {
917         case '<':
918         case SPECIAL_UNSIGNED_LT:
```

```
919                 if (sval_cmp(left->min, right->max) < 0)
920                         return 1;
921                 return 0;
922         case SPECIAL_UNSIGNED_LTE:
923         case SPECIAL_LTE:
924                 if (sval_cmp(left->min, right->max) <= 0)
925                         return 1;
926                 return 0;
927         case SPECIAL_EQUAL:
928                 if (sval_cmp(left->max, right->min) < 0)
929                         return 0;
930                 if (sval_cmp(left->min, right->max) > 0)
931                         return 0;
932                 return 1;
933         case SPECIAL_UNSIGNED_GTE:
934         case SPECIAL_GTE:
935                 if (sval_cmp(left->max, right->min) >= 0)
936                         return 1;
937                 return 0;
938         case '>':
939         case SPECIAL_UNSIGNED_GT:
940                 if (sval_cmp(left->max, right->min) > 0)
941                         return 1;
942                 return 0;
943         case SPECIAL_NOTEQUAL:
944                 if (sval_cmp(left->min, left->max) != 0)
945                         return 1;
946                 if (sval_cmp(right->min, right->max) != 0)
947                         return 1;
948                 if (sval_cmp(left->min, right->min) != 0)
949                         return 1;
950                 return 0;
951         default:
952                 sm_perror("unhandled comparison %d", comparison);
953                 return 0;
954         }
955         return 0;
956 }

958 int true_comparison_range_LR(int comparison, struct data_range *var, struct data
959 {
960         if (left)
961                 return true_comparison_range(var, comparison, val);
962         else
963                 return true_comparison_range(val, comparison, var);
964 }

966 static int false_comparison_range_sval(struct data_range *left, int comparison,
967 {
968         switch (comparison) {
969         case '<':
970         case SPECIAL_UNSIGNED_LT:
971                 if (sval_cmp(left->max, right->min) >= 0)
972                         return 1;
973                 return 0;
974         case SPECIAL_UNSIGNED_LTE:
975         case SPECIAL_LTE:
976                 if (sval_cmp(left->max, right->min) > 0)
977                         return 1;
978                 return 0;
979         case SPECIAL_EQUAL:
980                 if (sval_cmp(left->min, left->max) != 0)
981                         return 1;
982                 if (sval_cmp(right->min, right->max) != 0)
983                         return 1;
984                 if (sval_cmp(left->min, right->min) != 0)
```



```
985                         return 1;
986                 return 0;
987         case SPECIAL_UNSIGNED_GTE:
988         case SPECIAL_GTE:
989                 if (sval_cmp(left->min, right->max) < 0)
990                         return 1;
991                 return 0;
992         case '>':
993         case SPECIAL_UNSIGNED_GT:
994                 if (sval_cmp(left->min, right->max) <= 0)
995                         return 1;
996                 return 0;
997         case SPECIAL_NOTEQUAL:
998                 if (sval_cmp(left->max, right->min) < 0)
999                         return 0;
1000                if (sval_cmp(left->min, right->max) > 0)
1001                        return 0;
1002                return 1;
1003        default:
1004                sm_perror("unhandled comparison %d", comparison);
1005                return 0;
1006        }
1007        return 0;
1008 }

1010 int false_comparison_range_LR(int comparison, struct data_range *var, struct dat
1011 {
1012         if (left)
1013                 return false_comparison_range_sval(var, comparison, val);
1014         else
1015                 return false_comparison_range_sval(val, comparison, var);
1016 }

1018 int possibly_true(struct expression *left, int comparison, struct expression *ri
1019 {
1020         struct range_list *rl_left, *rl_right;
1021         struct data_range *tmp_left, *tmp_right;
1022         struct symbol *type;

1024         if (!get_implied_rl(left, &rl_left))
1025                 return 1;
1026         if (!get_implied_rl(right, &rl_right))
1027                 return 1;

1029         type = rl_type(rl_left);
1030         if (type_positive_bits(type) < type_positive_bits(rl_type(rl_right)))
1031                 type = rl_type(rl_right);
1032         if (type_positive_bits(type) < 31)
1033                 type = &int_ctype;

1035         rl_left = cast_rl(type, rl_left);
1036         rl_right = cast_rl(type, rl_right);

1038         FOR_EACH_PTR(rl_left, tmp_left) {
1039                 FOR_EACH_PTR(rl_right, tmp_right) {
1040                         if (true_comparison_range(tmp_left, comparison, tmp_righ
1041                                 return 1;
1042                 } END_FOR_EACH_PTR(tmp_right);
1043         } END_FOR_EACH_PTR(tmp_left);
1044         return 0;
1045 }

1047 int possibly_false(struct expression *left, int comparison, struct expression *r
1048 {
1049         struct range_list *rl_left, *rl_right;
1050         struct data_range *tmp_left, *tmp_right;
```

```
1051         struct symbol *type;

1053         if (!get_implied_rl(left, &rl_left))
1054                 return 1;
1055         if (!get_implied_rl(right, &rl_right))
1056                 return 1;

1058         type = rl_type(rl_left);
1059         if (type_positive_bits(type) < type_positive_bits(rl_type(rl_right)))
1060                 type = rl_type(rl_right);
1061         if (type_positive_bits(type) < 31)
1062                 type = &int_ctype;

1064         rl_left = cast_rl(type, rl_left);
1065         rl_right = cast_rl(type, rl_right);

1067         FOR_EACH_PTR(rl_left, tmp_left) {
1068                 FOR_EACH_PTR(rl_right, tmp_right) {
1069                         if (false_comparison_range_sval(tmp_left, comparison, tm
1070                                 return 1;
1071                 } END_FOR_EACH_PTR(tmp_right);
1072         } END_FOR_EACH_PTR(tmp_left);
1073         return 0;
1074 }

1076 int possibly_true_rl(struct range_list *left_ranges, int comparison, struct rang
1077 {
1078         struct data_range *left_tmp, *right_tmp;
1079         struct symbol *type;

1081         if (!left_ranges || !right_ranges)
1082                 return 1;

1084         type = rl_type(left_ranges);
1085         if (type_positive_bits(type) < type_positive_bits(rl_type(right_ranges))
1086                 type = rl_type(right_ranges);
1087         if (type_positive_bits(type) < 31)
1088                 type = &int_ctype;

1090         left_ranges = cast_rl(type, left_ranges);
1091         right_ranges = cast_rl(type, right_ranges);

1093         FOR_EACH_PTR(left_ranges, left_tmp) {
1094                 FOR_EACH_PTR(right_ranges, right_tmp) {
1095                         if (true_comparison_range(left_tmp, comparison, right_tm
1096                                 return 1;
1097                 } END_FOR_EACH_PTR(right_tmp);
1098         } END_FOR_EACH_PTR(left_tmp);
1099         return 0;
1100 }

1102 int possibly_false_rl(struct range_list *left_ranges, int comparison, struct ran
1103 {
1104         struct data_range *left_tmp, *right_tmp;
1105         struct symbol *type;

1107         if (!left_ranges || !right_ranges)
1108                 return 1;

1110         type = rl_type(left_ranges);
1111         if (type_positive_bits(type) < type_positive_bits(rl_type(right_ranges))
1112                 type = rl_type(right_ranges);
1113         if (type_positive_bits(type) < 31)
1114                 type = &int_ctype;

1116         left_ranges = cast_rl(type, left_ranges);
```



```
1117         right_ranges = cast_rl(type, right_ranges);

1119         FOR_EACH_PTR(left_ranges, left_tmp) {
1120                 FOR_EACH_PTR(right_ranges, right_tmp) {
1121                         if (false_comparison_range_sval(left_tmp, comparison, ri
1122                                 return 1;
1123                 } END_FOR_EACH_PTR(right_tmp);
1124         } END_FOR_EACH_PTR(left_tmp);
1125         return 0;
1126 }

1128 /* FIXME: the _rl here stands for right left so really it should be _lr */
1129 int possibly_true_rl_LR(int comparison, struct range_list *a, struct range_list
1130 {
1131         if (left)
1132                 return possibly_true_rl(a, comparison, b);
1133         else
1134                 return possibly_true_rl(b, comparison, a);
1135 }

1137 int possibly_false_rl_LR(int comparison, struct range_list *a, struct range_list
1138 {
1139         if (left)
1140                 return possibly_false_rl(a, comparison, b);
1141         else
1142                 return possibly_false_rl(b, comparison, a);
1143 }

1145 int rl_has_sval(struct range_list *rl, sval_t sval)
1146 {
1147         struct data_range *tmp;

1149         FOR_EACH_PTR(rl, tmp) {
1150                 if (sval_cmp(tmp->min, sval) <= 0 &&
1151                     sval_cmp(tmp->max, sval) >= 0)
1152                         return 1;
1153         } END_FOR_EACH_PTR(tmp);
1154         return 0;
1155 }

1157 void tack_on(struct range_list **list, struct data_range *drange)
1158 {
1159         add_ptr_list(list, drange);
1160 }

1162 void push_rl(struct range_list_stack **rl_stack, struct range_list *rl)
1163 {
1164         add_ptr_list(rl_stack, rl);
1165 }

1167 struct range_list *pop_rl(struct range_list_stack **rl_stack)
1168 {
1169         struct range_list *rl;

1171         rl = last_ptr_list((struct ptr_list *)*rl_stack);
1172         delete_ptr_list_last((struct ptr_list **)rl_stack);
1173         return rl;
1174 }

1176 struct range_list *top_rl(struct range_list_stack *rl_stack)
1177 {
1178         struct range_list *rl;

1180         rl = last_ptr_list((struct ptr_list *)rl_stack);
1181         return rl;
1182 }
```

```
1184 void filter_top_rl(struct range_list_stack **rl_stack, struct range_list *filter
1185 {
1186         struct range_list *rl;

1188         rl = pop_rl(rl_stack);
1189         rl = rl_filter(rl, filter);
1190         push_rl(rl_stack, rl);
1191 }

1193 struct range_list *rl_truncate_cast(struct symbol *type, struct range_list *rl)
1194 {
1195         struct data_range *tmp;
1196         struct range_list *ret = NULL;
1197         sval_t min, max;

1199         if (!rl)
1200                 return NULL;

1202         if (!type || type == rl_type(rl))
1203                 return rl;

1205         FOR_EACH_PTR(rl, tmp) {
1206                 min = tmp->min;
1207                 max = tmp->max;
1208                 if (type_bits(type) < type_bits(rl_type(rl))) {
1209                         min.uvalue = tmp->min.uvalue & ((1ULL << type_bits(type)
1210                         max.uvalue = tmp->max.uvalue & ((1ULL << type_bits(type)
1211                 }
1212                 if (sval_cmp(min, max) > 0) {
1213                         min = sval_cast(type, min);
1214                         max = sval_cast(type, max);
1215                 }
1216                 add_range_t(type, &ret, min, max);
1217         } END_FOR_EACH_PTR(tmp);

1219         return ret;
1220 }

1222 static int rl_is_sane(struct range_list *rl)
1223 {
1224         struct data_range *tmp;
1225         struct symbol *type;

1227         type = rl_type(rl);
1228         FOR_EACH_PTR(rl, tmp) {
1229                 if (!sval_fits(type, tmp->min))
1230                         return 0;
1231                 if (!sval_fits(type, tmp->max))
1232                         return 0;
1233                 if (sval_cmp(tmp->min, tmp->max) > 0)
1234                         return 0;
1235         } END_FOR_EACH_PTR(tmp);

1237         return 1;
1238 }

1240 static int rl_type_consistent(struct range_list *rl)
1241 {
1242         struct data_range *tmp;
1243         struct symbol *type;

1245         type = rl_type(rl);
1246         FOR_EACH_PTR(rl, tmp) {
1247                 if (type != tmp->min.type || type != tmp->max.type)
1248                         return 0;
```



```
1249         } END_FOR_EACH_PTR(tmp);
1250         return 1;
1251 }

1253 static struct range_list *cast_to_bool(struct range_list *rl)
1254 {
1255         struct data_range *tmp;
1256         struct range_list *ret = NULL;
1257         int has_one = 0;
1258         int has_zero = 0;
1259         sval_t min = { .type = &bool_ctype };
1260         sval_t max = { .type = &bool_ctype };

1262         FOR_EACH_PTR(rl, tmp) {
1263                 if (tmp->min.value || tmp->max.value)
1264                         has_one = 1;
1265                 if (sval_is_negative(tmp->min) &&
1266                     sval_is_negative(tmp->max))
1267                         continue;
1268                 if (tmp->min.value == 0 ||
1269                     tmp->max.value == 0)
1270                         has_zero = 1;
1271                 if (sval_is_negative(tmp->min) &&
1272                     tmp->max.value > 0)
1273                         has_zero = 1;
1274         } END_FOR_EACH_PTR(tmp);

1276         if (!has_zero)
1277                 min.value = 1;
1278         if (has_one)
1279                 max.value = 1;

1281         add_range(&ret, min, max);
1282         return ret;
1283 }

1285 struct range_list *cast_rl(struct symbol *type, struct range_list *rl)
1286 {
1287         struct data_range *tmp;
1288         struct range_list *ret = NULL;

1290         if (!rl)
1291                 return NULL;

1293         if (!type)
1294                 return rl;
1295         if (!rl_is_sane(rl))
1296                 return alloc_whole_rl(type);
1297         if (type == rl_type(rl) && rl_type_consistent(rl))
1298                 return rl;

1300         if (type == &bool_ctype)
1301                 return cast_to_bool(rl);

1303         FOR_EACH_PTR(rl, tmp) {
1304                 add_range_t(type, &ret, tmp->min, tmp->max);
1305         } END_FOR_EACH_PTR(tmp);

1307         if (!ret)
1308                 return alloc_whole_rl(type);

1310         return ret;
1311 }

1313 struct range_list *rl_invert(struct range_list *orig)
1314 {
```

```
1315         struct range_list *ret = NULL;
1316         struct data_range *tmp;
1317         sval_t gap_min, abs_max, sval;

1319         if (!orig)
1320                 return NULL;
1321         if (type_bits(rl_type(orig)) < 0)  /* void type mostly */
1322                 return NULL;

1324         gap_min = sval_type_min(rl_min(orig).type);
1325         abs_max = sval_type_max(rl_max(orig).type);

1327         FOR_EACH_PTR(orig, tmp) {
1328                 if (sval_cmp(tmp->min, gap_min) > 0) {
1329                         sval = sval_type_val(tmp->min.type, tmp->min.value - 1);
1330                         add_range(&ret, gap_min, sval);
1331                 }
1332                 if (sval_cmp(tmp->max, abs_max) == 0)
1333                         return ret;
1334                 gap_min = sval_type_val(tmp->max.type, tmp->max.value + 1);
1335         } END_FOR_EACH_PTR(tmp);

1337         if (sval_cmp(gap_min, abs_max) <= 0)
1338                 add_range(&ret, gap_min, abs_max);

1340         return ret;
1341 }

1343 struct range_list *rl_filter(struct range_list *rl, struct range_list *filter)
1344 {
1345         struct data_range *tmp;

1347         FOR_EACH_PTR(filter, tmp) {
1348                 rl = remove_range(rl, tmp->min, tmp->max);
1349         } END_FOR_EACH_PTR(tmp);

1351         return rl;
1352 }

1354 struct range_list *rl_intersection(struct range_list *one, struct range_list *tw
1355 {
1356         struct range_list *one_orig;
1357         struct range_list *two_orig;
1358         struct range_list *ret;
1359         struct symbol *ret_type;
1360         struct symbol *small_type;
1361         struct symbol *large_type;

1363         if (!two)
1364                 return NULL;
1365         if (!one)
1366                 return NULL;

1368         one_orig = one;
1369         two_orig = two;

1371         ret_type = rl_type(one);
1372         small_type = rl_type(one);
1373         large_type = rl_type(two);

1375         if (type_bits(rl_type(two)) < type_bits(small_type)) {
1376                 small_type = rl_type(two);
1377                 large_type = rl_type(one);
1378         }

1380         one = cast_rl(large_type, one);
```



```
1381         two = cast_rl(large_type, two);

1383         ret = one;
1384         one = rl_invert(one);
1385         two = rl_invert(two);

1387         ret = rl_filter(ret, one);
1388         ret = rl_filter(ret, two);

1390         one = cast_rl(small_type, one_orig);
1391         two = cast_rl(small_type, two_orig);

1393         one = rl_invert(one);
1394         two = rl_invert(two);

1396         ret = cast_rl(small_type, ret);
1397         ret = rl_filter(ret, one);
1398         ret = rl_filter(ret, two);

1400         return cast_rl(ret_type, ret);
1401 }

1403 static struct range_list *handle_mod_rl(struct range_list *left, struct range_li
1404 {
1405         sval_t zero;
1406         sval_t max;

1408         max = rl_max(right);
1409         if (sval_is_max(max))
1410                 return left;
1411         if (max.value == 0)
1412                 return NULL;
1413         max.value--;
1414         if (sval_is_negative(max))
1415                 return NULL;
1416         if (sval_cmp(rl_max(left), max) < 0)
1417                 return left;
1418         zero = max;
1419         zero.value = 0;
1420         return alloc_rl(zero, max);
1421 }

1423 static struct range_list *get_neg_rl(struct range_list *rl)
1424 {
1425         struct data_range *tmp;
1426         struct data_range *new;
1427         struct range_list *ret = NULL;

1429         if (!rl)
1430                 return NULL;
1431         if (sval_is_positive(rl_min(rl)))
1432                 return NULL;

1434         FOR_EACH_PTR(rl, tmp) {
1435                 if (sval_is_positive(tmp->min))
1436                         break;
1437                 if (sval_is_positive(tmp->max)) {
1438                         new = alloc_range(tmp->min, tmp->max);
1439                         new->max.value = -1;
1440                         add_range(&ret, new->min, new->max);
1441                         break;
1442                 }
1443                 add_range(&ret, tmp->min, tmp->max);
1444         } END_FOR_EACH_PTR(tmp);

1446         return ret;
```

```
}

static struct range_list *get_pos_rl(struct range_list *rl)
{
        struct data_range *tmp;
        struct data_range *new;
        struct range_list *ret = NULL;

        if (!rl)
                return NULL;
        if (sval_is_negative(rl_max(rl)))
                return NULL;

        FOR_EACH_PTR(rl, tmp) {
                if (sval_is_negative(tmp->max))
                        continue;
                if (sval_is_positive(tmp->min)) {
                        add_range(&ret, tmp->min, tmp->max);
                        continue;
                }
                new = alloc_range(tmp->min, tmp->max);
                new->min.value = 0;
                add_range(&ret, new->min, new->max);
        } END_FOR_EACH_PTR(tmp);

        return ret;
}

static struct range_list *divide_rl_helper(struct range_list *left, struct range
{
        sval_t right_min, right_max;
        sval_t min, max;

        if (!left || !right)
                return NULL;

        /* let's assume we never divide by zero */
        right_min = rl_min(right);
        right_max = rl_max(right);
        if (right_min.value == 0 && right_max.value == 0)
                return NULL;
        if (right_min.value == 0)
                right_min.value = 1;
        if (right_max.value == 0)
                right_max.value = -1;

        max = sval_binop(rl_max(left), '/', right_min);
        min = sval_binop(rl_min(left), '/', right_max);

        return alloc_rl(min, max);
}

static struct range_list *handle_divide_rl(struct range_list *left, struct range
{
        struct range_list *left_neg, *left_pos, *right_neg, *right_pos;
        struct range_list *neg_neg, *neg_pos, *pos_neg, *pos_pos;
        struct range_list *ret;

        if (is_whole_rl(right))
                return NULL;

        left_neg = get_neg_rl(left);
        left_pos = get_pos_rl(left);
        right_neg = get_neg_rl(right);
        right_pos = get_pos_rl(right);
```



```
        neg_neg = divide_rl_helper(left_neg, right_neg);
        neg_pos = divide_rl_helper(left_neg, right_pos);
        pos_neg = divide_rl_helper(left_pos, right_neg);
        pos_pos = divide_rl_helper(left_pos, right_pos);

        ret = rl_union(neg_neg, neg_pos);
        ret = rl_union(ret, pos_neg);
        return rl_union(ret, pos_pos);
}

static struct range_list *handle_add_mult_rl(struct range_list *left, int op, st
{
        sval_t min, max;

        if (sval_binop_overflows(rl_min(left), op, rl_min(right)))
                return NULL;
        min = sval_binop(rl_min(left), op, rl_min(right));

        if (sval_binop_overflows(rl_max(left), op, rl_max(right)))
                return NULL;
        max = sval_binop(rl_max(left), op, rl_max(right));

        return alloc_rl(min, max);
}

static struct range_list *handle_sub_rl(struct range_list *left_orig, struct ran
{
        struct range_list *left_rl, *right_rl;
        struct symbol *type;
        sval_t min, max;
        sval_t min_ll, max_ll, res_ll;
        sval_t tmp;

        /* TODO:  These things should totally be using dranges where possible */

        if (!left_orig || !right_orig)
                return NULL;

        type = &int_ctype;
        if (type_positive_bits(rl_type(left_orig)) > type_positive_bits(type))
                type = rl_type(left_orig);
        if (type_positive_bits(rl_type(right_orig)) > type_positive_bits(type))
                type = rl_type(right_orig);

        left_rl = cast_rl(type, left_orig);
        right_rl = cast_rl(type, right_orig);

        max = rl_max(left_rl);
        min = sval_type_min(type);

        min_ll = rl_min(left_rl);
        min_ll.type = &llong_ctype;
        max_ll = rl_max(right_rl);
        max_ll.type = &llong_ctype;
        res_ll = min_ll;
        res_ll.value = min_ll.value - max_ll.value;

        if (!sval_binop_overflows(rl_min(left_rl), '-', rl_max(right_rl))) {
                tmp = sval_binop(rl_min(left_rl), '-', rl_max(right_rl));
                if (sval_cmp(tmp, min) > 0)
                        min = tmp;
        } else if (type_positive_bits(type) < 63 &&
                   !sval_binop_overflows(min_ll, '-', max_ll) &&
                   (min.value != 0 && sval_cmp(res_ll, min) >= 0)) {
                struct range_list *left_casted, *right_casted, *result;
```

```
1579                 left_casted = cast_rl(&llong_ctype, left_orig);
1580                 right_casted = cast_rl(&llong_ctype, right_orig);
1581                 result = handle_sub_rl(left_casted, right_casted);
1582                 return cast_rl(type, result);
1583         }

1585         if (!sval_is_max(rl_max(left_rl))) {
1586                 tmp = sval_binop(rl_max(left_rl), '-', rl_min(right_rl));
1587                 if (sval_cmp(tmp, max) < 0)
1588                         max = tmp;
1589         }

1591         if (sval_is_min(min) && sval_is_max(max))
1592                 return NULL;

1594         return alloc_rl(min, max);
1595 }

1597 static unsigned long long rl_bits_always_set(struct range_list *rl)
1598 {
1599         return sval_fls_mask(rl_min(rl));
1600 }

1602 static unsigned long long rl_bits_maybe_set(struct range_list *rl)
1603 {
1604         return sval_fls_mask(rl_max(rl));
1605 }

1607 static struct range_list *handle_OR_rl(struct range_list *left, struct range_lis
1608 {
1609         unsigned long long left_min, left_max, right_min, right_max;
1610         sval_t min, max;
1611         sval_t sval;

1613         if ((rl_to_sval(left, &sval) || rl_to_sval(right, &sval)) &&
1614             !sval_binop_overflows(rl_max(left), '+', rl_max(right)))
1615                 return rl_binop(left, '+', right);

1617         left_min = rl_bits_always_set(left);
1618         left_max = rl_bits_maybe_set(left);
1619         right_min = rl_bits_always_set(right);
1620         right_max = rl_bits_maybe_set(right);

1622         min.type = max.type = &ullong_ctype;
1623         min.uvalue = left_min | right_min;
1624         max.uvalue = left_max | right_max;

1626         return cast_rl(rl_type(left), alloc_rl(min, max));
1627 }

1629 static struct range_list *handle_XOR_rl(struct range_list *left, struct range_li
1630 {
1631         unsigned long long left_set, left_maybe;
1632         unsigned long long right_set, right_maybe;
1633         sval_t zero, max;

1635         left_set = rl_bits_always_set(left);
1636         left_maybe = rl_bits_maybe_set(left);

1638         right_set = rl_bits_always_set(right);
1639         right_maybe = rl_bits_maybe_set(right);

1641         zero = max = rl_min(left);
1642         zero.uvalue = 0;
1643         max.uvalue = fls_mask((left_maybe | right_maybe) ^ (left_set & right_set
```



```
1645         return cast_rl(rl_type(left), alloc_rl(zero, max));
1646 }

1648 static struct range_list *handle_AND_rl(struct range_list *left, struct range_li
1649 {
1650         unsigned long long left_set, left_maybe;
1651         unsigned long long right_set, right_maybe;
1652         sval_t zero, max;

1654         return NULL;

1656         left_set = rl_bits_always_set(left);
1657         left_maybe = rl_bits_maybe_set(left);

1659         right_set = rl_bits_always_set(right);
1660         right_maybe = rl_bits_maybe_set(right);

1662         zero = max = rl_min(left);
1663         zero.uvalue = 0;
1664         max.uvalue = fls_mask((left_maybe | right_maybe) ^ (left_set & right_set

1666         return cast_rl(rl_type(left), alloc_rl(zero, max));
1667 }

1669 struct range_list *rl_binop(struct range_list *left, int op, struct range_list *
1670 {
1671         struct symbol *cast_type;
1672         sval_t left_sval, right_sval;
1673         struct range_list *ret = NULL;

1675         cast_type = rl_type(left);
1676         if (sval_type_max(rl_type(left)).uvalue < sval_type_max(rl_type(right)).
1677                 cast_type = rl_type(right);
1678         if (sval_type_max(cast_type).uvalue < INT_MAX)
1679                 cast_type = &int_ctype;

1681         left = cast_rl(cast_type, left);
1682         right = cast_rl(cast_type, right);

1684         if (!left && !right)
1685                 return NULL;

1687         if (rl_to_sval(left, &left_sval) && rl_to_sval(right, &right_sval)) {
1688                 sval_t val = sval_binop(left_sval, op, right_sval);
1689                 return alloc_rl(val, val);
1690         }

1692         switch (op) {
1693         case '%':
1694                 ret = handle_mod_rl(left, right);
1695                 break;
1696         case '/':
1697                 ret = handle_divide_rl(left, right);
1698                 break;
1699         case '*':
1700         case '+':
1701                 ret = handle_add_mult_rl(left, op, right);
1702                 break;
1703         case '|':
1704                 ret = handle_OR_rl(left, right);
1705                 break;
1706         case '^':
1707                 ret = handle_XOR_rl(left, right);
1708                 break;
1709         case '&':
1710                 ret = handle_AND_rl(left, right);
```

```
1711                    break;
1712            case '-':
1713                    ret = handle_sub_rl(left, right);
1714                    break;
1715            /* FIXME:  Do the rest as well */
1716            case SPECIAL_RIGHTSHIFT:
1717            case SPECIAL_LEFTSHIFT:
1718                    break;
1719            }

1721            return ret;
1722    }

1724    void free_data_info_allocs(void)
1725    {
1726            struct allocator_struct *desc = &data_info_allocator;
1727            struct allocation_blob *blob = desc->blobs;

1729            free_all_rl();
1730            clear_math_cache();

1732            desc->blobs = NULL;
1733            desc->allocations = 0;
1734            desc->total_bytes = 0;
1735            desc->useful_bytes = 0;
1736            desc->freelist = NULL;
1737            while (blob) {
1738                    struct allocation_blob *next = blob->next;
1739                    blob_free(blob, desc->chunking);
1740                    blob = next;
1741            }
1742            clear_data_range_alloc();
1743    }

1745    void split_comparison_rl(struct range_list *left_orig, int op, struct range_list
1746                    struct range_list **left_true_rl, struct range_list **left_false
1747                    struct range_list **right_true_rl, struct range_list **right_fal
1748    {
1749            struct range_list *left_true, *left_false;
1750            struct range_list *right_true, *right_false;
1751            sval_t min, max;

1753            min = sval_type_min(rl_type(left_orig));
1754            max = sval_type_max(rl_type(left_orig));

1756            left_true = clone_rl(left_orig);
1757            left_false = clone_rl(left_orig);
1758            right_true = clone_rl(right_orig);
1759            right_false = clone_rl(right_orig);

1761            switch (op) {
1762            case '<':
1763            case SPECIAL_UNSIGNED_LT:
1764                    left_true = remove_range(left_orig, rl_max(right_orig), max);
1765                    if (!sval_is_min(rl_min(right_orig))) {
1766                            left_false = remove_range(left_orig, min, sub_one(rl_min
1767                    }

1769                    right_true = remove_range(right_orig, min, rl_min(left_orig));
1770                    if (!sval_is_max(rl_max(left_orig)))
1771                            right_false = remove_range(right_orig, add_one(rl_max(le
1772                    break;
1773            case SPECIAL_UNSIGNED_LTE:
1774            case SPECIAL_LTE:
1775                    if (!sval_is_max(rl_max(right_orig)))
1776                            left_true = remove_range(left_orig, add_one(rl_max(right
```



```
1777                    left_false = remove_range(left_orig, min, rl_min(right_orig));

1779                    if (!sval_is_min(rl_min(left_orig)))
1780                            right_true = remove_range(right_orig, min, sub_one(rl_mi
1781                    right_false = remove_range(right_orig, rl_max(left_orig), max);

1783                    if (sval_cmp(rl_min(left_orig), rl_min(right_orig)) == 0)
1784                            left_false = remove_range(left_false, rl_min(left_orig),
1785                    if (sval_cmp(rl_max(left_orig), rl_max(right_orig)) == 0)
1786                            right_false = remove_range(right_false, rl_max(left_orig
1787                    break;
1788            case SPECIAL_EQUAL:
1789                    left_true = rl_intersection(left_orig, right_orig);
1790                    right_true = clone_rl(left_true);

1792                    if (sval_cmp(rl_min(right_orig), rl_max(right_orig)) == 0)
1793                            left_false = remove_range(left_orig, rl_min(right_orig),
1794                    if (sval_cmp(rl_min(left_orig), rl_max(left_orig)) == 0)
1795                            right_false = remove_range(right_orig, rl_min(left_orig)
1796                    break;
1797            case SPECIAL_UNSIGNED_GTE:
1798            case SPECIAL_GTE:
1799                    if (!sval_is_min(rl_min(right_orig)))
1800                            left_true = remove_range(left_orig, min, sub_one(rl_min(
1801                    left_false = remove_range(left_orig, rl_max(right_orig), max);

1803                    if (!sval_is_max(rl_max(left_orig)))
1804                            right_true = remove_range(right_orig, add_one(rl_max(lef
1805                    right_false = remove_range(right_orig, min, rl_min(left_orig));

1807                    if (sval_cmp(rl_min(left_orig), rl_min(right_orig)) == 0)
1808                            right_false = remove_range(right_false, rl_min(left_orig
1809                    if (sval_cmp(rl_max(left_orig), rl_max(right_orig)) == 0)
1810                            left_false = remove_range(left_false, rl_max(left_orig),
1811                    break;
1812            case '>':
1813            case SPECIAL_UNSIGNED_GT:
1814                    left_true = remove_range(left_orig, min, rl_min(right_orig));
1815                    if (!sval_is_max(rl_max(right_orig)))
1816                            left_false = remove_range(left_orig, add_one(rl_max(righ

1818                    right_true = remove_range(right_orig, rl_max(left_orig), max);
1819                    if (!sval_is_min(rl_min(left_orig)))
1820                            right_false = remove_range(right_orig, min, sub_one(rl_m
1821                    break;
1822            case SPECIAL_NOTEQUAL:
1823                    left_false = rl_intersection(left_orig, right_orig);
1824                    right_false = clone_rl(left_false);

1826                    if (sval_cmp(rl_min(right_orig), rl_max(right_orig)) == 0)
1827                            left_true = remove_range(left_orig, rl_min(right_orig),
1828                    if (sval_cmp(rl_min(left_orig), rl_max(left_orig)) == 0)
1829                            right_true = remove_range(right_orig, rl_min(left_orig),
1830                    break;
1831            default:
1832                    sm_perror(" unhandled comparison %d", op);
1833                    return;
1834            }

1836            if (left_true_rl) {
1837                    *left_true_rl = left_true;
1838                    *left_false_rl = left_false;
1839            }
1840            if (right_true_rl) {
1841                    *right_true_rl = right_true;
1842                    *right_false_rl = right_false;
```

```
1843         }
1844 }
```

```
**********************************************************
    3623 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_real_absolute.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Say we have a line like:
 * foo = bar / 8;
 * Assume we don't know anything about bar.  Well, now we know that foo is less
 * than UINT_MAX / 8.  Which might be useful, but it probably is misleading
 * useless knowledge.  Up to now we have ignored those but now we have said to
 * store them.
 *
 * It also works if you have something like "foo = (int)(char)unknown_var;".
 *
 * I feel like this data doesn't have to be perfect, it just has to be better
 * than nothing and that will help eliminate some false positives.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static void pre_merge_hook(struct sm_state *sm)
{
        struct smatch_state *abs;
        struct smatch_state *extra;
        struct range_list *rl;

        extra = get_state(SMATCH_EXTRA, sm->name, sm->sym);
        if (!extra || !estate_rl(extra))
                return;
        abs = get_state(my_id, sm->name, sm->sym);
        if (!abs || !estate_rl(abs)) {
                set_state(my_id, sm->name, sm->sym, clone_estate(extra));
                return;
        }
        rl = rl_intersection(estate_rl(abs), estate_rl(extra));
        set_state(my_id, sm->name, sm->sym, alloc_estate_rl(clone_rl(rl)));
}

static struct smatch_state *empty_state(struct sm_state *sm)
{
        return alloc_estate_empty();
}
```



```
static void reset(struct sm_state *sm, struct expression *mod_expr)
{
        set_state(my_id, sm->name, sm->sym, alloc_estate_whole(estate_type(sm->s
}

static int in_iterator_pre_statement(void)
{
        struct statement *stmt;

        /*
         * we can't use __cur_stmt because that isn't set for
         * iterator_pre_statement.  Kind of a mess.
         *
         */

        stmt = last_ptr_list((struct ptr_list *)big_statement_stack);

        if (!stmt || !stmt->parent)
                return 0;
        if (stmt->parent->type != STMT_ITERATOR)
                return 0;
        if (stmt->parent->iterator_pre_statement != stmt)
                return 0;
        return 1;
}

static void match_assign(struct expression *expr)
{
        struct range_list *rl;
        struct symbol *type;
        sval_t sval;

        if (expr->op != '=')
                return;
        if (is_fake_call(expr->right))
                return;
        if (in_iterator_pre_statement())
                return;

        get_real_absolute_rl(expr->right, &rl);

        type = get_type(expr->left);
        if (!type)
                return;

        rl = cast_rl(type, rl);
        if (is_whole_rl(rl) && !get_state_expr(my_id, expr->left))
                return;
        /* These are handled by smatch_extra.c */
        if (rl_to_sval(rl, &sval) && !get_state_expr(my_id, expr->left))
                return;

        set_state_expr(my_id, expr->left, alloc_estate_rl(clone_rl(rl)));
}

struct smatch_state *get_real_absolute_state(struct expression *expr)
{
        return get_state_expr(my_id, expr);
}

struct smatch_state *get_real_absolute_state_var_sym(const char *name, struct sy
{
        return get_state(my_id, name, sym);
}
```

```
127 void register_real_absolute(int id)
128 {
129         my_id = id;
130
131         add_pre_merge_hook(my_id, &pre_merge_hook);
132         add_unmatched_state_hook(my_id, &empty_state);
133         add_merge_hook(my_id, &merge_estates);
134         add_modification_hook(my_id, &reset);
135
136         add_hook(&match_assign, ASSIGNMENT_HOOK);
137 }
```

```
**********************************************************
    4187 Fri Dec 21 15:00:28 2018
new/usr/src/tools/smatch/src/smatch_recurse.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

#define RECURSE_LIMIT 10

static int recurse(struct expression *expr,
                   int (func)(struct expression *expr, void *p),
                   void *param, int nr)
{
        int ret;

        if (!expr)
                return 0;

        ret = func(expr, param);
        if (ret)
                return ret;

        if (nr > RECURSE_LIMIT)
                return -1;
        nr++;

        switch (expr->type) {
        case EXPR_PREOP:
                ret = recurse(expr->unop, func, param, nr);
                break;
        case EXPR_POSTOP:
                ret = recurse(expr->unop, func, param, nr);
                break;
        case EXPR_STATEMENT:
                return -1;
                break;
        case EXPR_LOGICAL:
        case EXPR_COMPARE:
        case EXPR_BINOP:
        case EXPR_COMMA:
                ret = recurse(expr->left, func, param, nr);
                if (ret)
                        return ret;
                ret = recurse(expr->right, func, param, nr);
                break;
        case EXPR_ASSIGNMENT:
                ret = recurse(expr->right, func, param, nr);
                if (ret)
```



```
                        return ret;
                ret = recurse(expr->left, func, param, nr);
                break;
        case EXPR_DEREF:
                ret = recurse(expr->deref, func, param, nr);
                break;
        case EXPR_SLICE:
                ret = recurse(expr->base, func, param, nr);
                break;
        case EXPR_CAST:
        case EXPR_FORCE_CAST:
                ret = recurse(expr->cast_expression, func, param, nr);
                break;
        case EXPR_SIZEOF:
        case EXPR_OFFSETOF:
        case EXPR_ALIGNOF:
                break;
        case EXPR_CONDITIONAL:
        case EXPR_SELECT:
                ret = recurse(expr->conditional, func, param, nr);
                if (ret)
                        return ret;
                ret = recurse(expr->cond_true, func, param, nr);
                if (ret)
                        return ret;
                ret = recurse(expr->cond_false, func, param, nr);
                break;
        case EXPR_CALL:
                return -1;
                break;
        case EXPR_INITIALIZER:
                return -1;
                break;
        case EXPR_IDENTIFIER:
                ret = recurse(expr->ident_expression, func, param, nr);
                break;
        case EXPR_INDEX:
                ret = recurse(expr->idx_expression, func, param, nr);
                break;
        case EXPR_POS:
                ret = recurse(expr->init_expr, func, param, nr);
                break;
        case EXPR_SYMBOL:
        case EXPR_STRING:
        case EXPR_VALUE:
                break;
        default:
                return -1;
                break;
        };
        return ret;
}

static int has_symbol_helper(struct expression *expr, void *_sym)
{
        struct symbol *sym = _sym;

        if (!expr || expr->type != EXPR_SYMBOL)
                return 0;
        if (expr->symbol == sym)
                return 1;
        return 0;
}

int has_symbol(struct expression *expr, struct symbol *sym)
{
```

```
127         return recurse(expr, has_symbol_helper, sym, 0);
128 }

130 struct expr_name_sym {
131         struct expression *expr;
132         char *name;
133         struct symbol *sym;
134 };

136 static int has_var_helper(struct expression *expr, void *_var)
137 {
138         struct expr_name_sym *xns = _var;
139         char *name;
140         struct symbol *sym;
141         int matched = 0;

143         if (!expr)
144                 return 0;
145         if (expr->type != xns->expr->type)
146                 return 0;
147         // I hope this is defined for everything?  It should work, right?
148         if (expr->op != xns->expr->op)
149                 return 0;

151         name = expr_to_var_sym(expr, &sym);
152         if (!name || !sym)
153                 goto free;
154         if (sym == xns->sym && strcmp(name, xns->name) == 0)
155                 matched = 1;
156 free:
157         free_string(name);
158         return matched;
159 }

161 int has_variable(struct expression *expr, struct expression *var)
162 {
163         struct expr_name_sym xns;
164         int ret = -1;

166         xns.expr = var;
167         xns.name = expr_to_var_sym(var, &xns.sym);
168         if (!xns.name || !xns.sym)
169                 goto free;
170         ret = recurse(expr, has_var_helper, &xns, 0);
171 free:
172         free_string(xns.name);
173         return ret;
174 }

176 static int has_inc_dec_helper(struct expression *expr, void *unused)
177 {
178         if (!expr)
179                 return 0;
180         if (expr->type != EXPR_PREOP && expr->type != EXPR_POSTOP)
181                 return 0;
182         if (expr->op == SPECIAL_INCREMENT || expr->op == SPECIAL_DECREMENT)
183                 return 1;
184         return 0;
185 }

187 int has_inc_dec(struct expression *expr)
188 {
189         return recurse(expr, has_inc_dec_helper, NULL, 0);
190 }
```

```
**********************************************************
    7281 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_return_to_param.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2017 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This is for smatch_extra.c to use.  It sort of like check_assigned_expr.c but
 * more limited.  Say a function returns "64min-s64max[$0->data]" and the caller
 * does "struct whatever *p = get_data(dev);" then we want to record that p is
 * now the same as "dev->data".  Then if we update "p->foo" it means we can
 * update "dev->data->foo" as well.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

extern int check_assigned_expr_id;
static int my_id;
static int link_id;

static struct smatch_state *alloc_my_state(const char *name, struct symbol *sym)
{
        struct smatch_state *state;

        state = __alloc_smatch_state(0);
        state->name = alloc_sname(name);
        state->data = sym;
        return state;
}

static void undef(struct sm_state *sm, struct expression *mod_expr)
{
        if (__in_fake_parameter_assign)
                return;
        set_state(my_id, sm->name, sm->sym, &undefined);
}

char *map_call_to_other_name_sym(const char *name, struct symbol *sym, struct sy
{
        struct smatch_state *state;
        int skip;
        char buf[256];

        /* skip 'foo->'.  This was checked in the caller. */
        skip = strlen(sym->ident->name) + 2;
```



```
        state = get_state(my_id, sym->ident->name, sym);
        if (!state || !state->data)
                return NULL;

        snprintf(buf, sizeof(buf), "%s->%s", state->name, name + skip);
        *new_sym = state->data;
        return alloc_string(buf);
}

static char *map_my_state_long_to_short(struct sm_state *sm, const char *name, s
{
        int len;
        char buf[256];

        if (sm->state->data != sym)
                return NULL;
        len = strlen(sm->state->name);
        if (strncmp(name, sm->state->name, len) != 0)
                return NULL;

        if (name[len] == '.')
                return NULL;
        if (!stack && name[len] != '-')
                return NULL;
        snprintf(buf, sizeof(buf), "%s%s", sm->name, name + len);
        *new_sym = sm->sym;
        return alloc_string(buf);
}

static char *map_assignment_long_to_short(struct sm_state *sm, const char *name,
{
        struct expression *orig_expr;
        struct symbol *orig_sym;
        int len;
        char buf[256];

        orig_expr = sm->state->data;
        if (!orig_expr)
                return NULL;

        /*
         * Say we have an assignment like:
         *     foo->bar->my_ptr = my_ptr;
         * We still expect the function to carry on using "my_ptr" as the
         * shorter name.  That's not a long to short mapping.
         *
         */
        if (orig_expr->type == EXPR_SYMBOL)
                return NULL;

        orig_sym = expr_to_sym(orig_expr);
        if (!orig_sym)
                return NULL;
        if (sym != orig_sym)
                return NULL;

        len = strlen(sm->state->name);
        if (strncmp(name, sm->state->name, len) != 0)
                return NULL;

        if (name[len] == '.')
                return NULL;
        if (!stack && name[len] != '-')
                return NULL;
        snprintf(buf, sizeof(buf), "%s%s", sm->name, name + len);
        *new_sym = sm->sym;
```

```
127         return alloc_string(buf);
128 }

130 /*
131  * Normally, we expect people to consistently refer to variables by the shortest
132  * name.  So they use "b->a" instead of "foo->bar.a" when both point to the
133  * same memory location.  However, when we're dealing across function boundaries
134  * then sometimes we pass frob(foo) which sets foo->bar.a.  In that case, we
135  * translate it to the shorter name.  Smatch extra updates the shorter name,
136  * which in turn updates the longer name.
137  *
138  */
139 static char *map_long_to_short_name_sym_helper(const char *name, struct symbol *
140 {
141         char *ret;
142         struct sm_state *sm;

144         *new_sym = NULL;

146         FOR_EACH_SM(__get_cur_stree(), sm) {
147                 if (sm->owner == my_id) {
148                         ret = map_my_state_long_to_short(sm, name, sym, new_sym,
149                         if (ret)
150                                 return ret;
151                         continue;
152                 }
153                 if (sm->owner == check_assigned_expr_id) {
154                         ret = map_assignment_long_to_short(sm, name, sym, new_sy
155                         if (ret)
156                                 return ret;
157                         continue;
158                 }
159         } END_FOR_EACH_SM(sm);

161         return NULL;
162 }

164 char *map_long_to_short_name_sym(const char *name, struct symbol *sym, struct sy
165 {
166         return map_long_to_short_name_sym_helper(name, sym, new_sym, 1);
167 }

169 char *map_long_to_short_name_sym_nostack(const char *name, struct symbol *sym, s
170 {
171         return map_long_to_short_name_sym_helper(name, sym, new_sym, 0);
172 }

174 char *map_call_to_param_name_sym(struct expression *expr, struct symbol **sym)
175 {
176         char *name;
177         struct symbol *start_sym;
178         struct smatch_state *state;

180         *sym = NULL;

182         name = expr_to_str_sym(expr, &start_sym);
183         if (!name)
184                 return NULL;
185         if (expr->type == EXPR_CALL)
186                 start_sym = expr_to_sym(expr->fn);

188         state = get_state(my_id, name, start_sym);
189         free_string(name);
190         if (!state || !state->data)
191                 return NULL;
```



```
193         *sym = state->data;
194         return alloc_string(state->name);
195 }

197 static void store_mapping_helper(char *left_name, struct symbol *left_sym, struc
198 {
199         const char *p = return_string;
200         char *close;
201         int param;
202         struct expression *arg, *new;
203         char *right_name;
204         struct symbol *right_sym;
205         char buf[256];

207         while (*p && *p != '[')
208                 p++;
209         if (!*p)
210                 return;
211         p++;
212         if (*p != '$')
213                 return;

215         snprintf(buf, sizeof(buf), "%s", p);
216         close = strchr(buf, ']');
217         if (!close)
218                 return;
219         *close = '\0';

221         param = atoi(buf + 1);
222         arg = get_argument_from_call_expr(call->args, param);
223         if (!arg)
224                 return;

226         new = gen_expression_from_key(arg, buf);
227         if (!new)
228                 return;

230         right_name = expr_to_var_sym(new, &right_sym);
231         if (!right_name || !right_sym)
232                 goto free;

234         set_state(my_id, left_name, left_sym, alloc_my_state(right_name, right_s
235         store_link(link_id, right_name, right_sym, left_name, left_sym);

237 free:
238         free_string(right_name);
239 }

241 void __add_return_to_param_mapping(struct expression *expr, const char *return_s
242 {
243         struct expression *call;
244         char *left_name = NULL;
245         struct symbol *left_sym;

247         if (expr->type == EXPR_ASSIGNMENT) {
248                 left_name = expr_to_var_sym(expr->left, &left_sym);
249                 if (!left_name || !left_sym)
250                         goto free;

252                 call = strip_expr(expr->right);
253                 if (call->type != EXPR_CALL)
254                         goto free;

256                 store_mapping_helper(left_name, left_sym, call, return_string);
257                 goto free;
258         }
```

```
260         if (expr->type == EXPR_CALL &&
261             expr_get_parent_stmt(expr) &&
262             expr_get_parent_stmt(expr)->type == STMT_RETURN) {
263                 call = strip_expr(expr);
264                 left_sym = expr_to_sym(call->fn);
265                 if (!left_sym)
266                         return;
267                 left_name = expr_to_str(call);
268                 if (!left_name)
269                         return;
271                 store_mapping_helper(left_name, left_sym, call, return_string);
272                 goto free;
274         }
276 free:
277         free_string(left_name);
278 }
280 void register_return_to_param(int id)
281 {
282         my_id = id;
283         add_modification_hook(my_id, &undef);
284 }
286 void register_return_to_param_links(int id)
287 {
288         link_id = id;
289         set_up_link_functions(my_id, link_id);
290 }
```

```
**********************************************************
    3723 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_returns.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2011 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

int RETURN_ID;

struct return_states_callback {
        void (*callback)(void);
};
ALLOCATOR(return_states_callback, "return states callbacks");
DECLARE_PTR_LIST(callback_list, struct return_states_callback);
static struct callback_list *callback_list;

DECLARE_PTR_LIST(stree_stack_stack, struct stree_stack);
static void push_stree_stack(struct stree_stack_stack **stack_stack, struct stre
{
        add_ptr_list(stack_stack, stack);
}

static struct stree_stack *pop_stree_stack(struct stree_stack_stack **stack_stac
{
        struct stree_stack *stack;

        stack = last_ptr_list((struct ptr_list *)*stack_stack);
        delete_ptr_list_last((struct ptr_list **)stack_stack);
        return stack;
}

static struct stree_stack *return_stree_stack;
static struct stree_stack_stack *saved_stack_stack;
static struct stree *all_return_states;
static struct stree_stack *saved_stack;

void all_return_states_hook(void (*callback)(void))
{
        struct return_states_callback *rs_cb = __alloc_return_states_callback(0)

        rs_cb->callback = callback;
        add_ptr_list(&callback_list, rs_cb);
}

static void call_hooks(void)
{
```



```
        struct return_states_callback *rs_cb;

        __set_fake_cur_stree_fast(all_return_states);
        FOR_EACH_PTR(callback_list, rs_cb) {
                rs_cb->callback();
        } END_FOR_EACH_PTR(rs_cb);
        __pop_fake_cur_stree_fast();
}

static void match_return(int return_id, char *return_ranges, struct expression *
{
        struct stree *stree;

        stree = clone_stree(__get_cur_stree());
        merge_stree_no_pools(&all_return_states, stree);
        push_stree(&return_stree_stack, stree);
}

static void match_end_func(struct symbol *sym)
{
        /*
         * FIXME: either this isn't needed or we need to copy a stree into the
         * return_stree_stack as well.
         */
        merge_stree(&all_return_states, __get_cur_stree());
        call_hooks();
}

static void match_save_states(struct expression *expr)
{
        push_stree(&saved_stack, all_return_states);
        all_return_states = NULL;

        push_stree_stack(&saved_stack_stack, return_stree_stack);
        return_stree_stack = NULL;
}

static void match_restore_states(struct expression *expr)
{
        /* This free_stree() isn't needed is it?? */
        free_stree(&all_return_states);

        all_return_states = pop_stree(&saved_stack);
        return_stree_stack = pop_stree_stack(&saved_stack_stack);
}

struct stree *get_all_return_states(void)
{
        return all_return_states;
}

struct stree_stack *get_all_return_strees(void)
{
        return return_stree_stack;
}

static void free_resources(struct symbol *sym)
{
        struct stree *tmp;

        free_stree(&all_return_states);

        FOR_EACH_PTR(return_stree_stack, tmp) {
                free_stree(&tmp);
        } END_FOR_EACH_PTR(tmp);
        free_stree_stack(&return_stree_stack);
```

```
127 }

129 void register_returns_early(int id)
130 {
131         RETURN_ID = id;

133         add_split_return_callback(match_return);
134 }

136 void register_returns(int id)
137 {
138         add_hook(&match_end_func, END_FUNC_HOOK);
139         add_hook(&match_save_states, INLINE_FN_START);
140         add_hook(&match_restore_states, INLINE_FN_END);
141         add_hook(&free_resources, AFTER_FUNC_HOOK);
142 }
```

```
**********************************************************
    2537 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scope.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2016 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

static struct statement_list *stmt_list;

static int end_of_function(struct statement *stmt)
{
        struct symbol *fn = get_base_type(cur_func_sym);

        /* err on the conservative side of things */
        if (!fn)
                return 1;
        if (stmt == fn->stmt || stmt == fn->inline_stmt)
                return 1;
        return 0;
}

/*
 * We're wasting a lot of time worrying about out of scope variables.
 * When we come to the end of a scope then just delete them all the out of
 * scope states.
 */
static void match_end_of_block(struct statement *stmt)
{
        struct statement *tmp;
        struct symbol *sym;

        if (end_of_function(stmt))
                return;

        FOR_EACH_PTR(stmt->stmts, tmp) {
                if (tmp->type != STMT_DECLARATION)
                        return;

                FOR_EACH_PTR(tmp->declaration, sym) {
                        if (!sym->ident)
                                continue;
                        __delete_all_states_sym(sym);
                } END_FOR_EACH_PTR(sym);
        } END_FOR_EACH_PTR(tmp);
}

static int is_outer_stmt(struct statement *stmt)
{
```



```
        struct symbol *fn;

        if (!cur_func_sym)
                return 0;
        fn = get_base_type(cur_func_sym);
        if (!fn)
                return 0;
        /*
         * There are times when ->parent is not set but it's set for
         * the outer statement so ignoring NULLs works as a work-around.
         */
        if (!stmt->parent)
                return 0;
        if (stmt->parent == fn->stmt ||
            stmt->parent == fn->inline_stmt)
                return 1;
        return 0;
}

static void match_stmt(struct statement *stmt)
{
        struct statement *tmp;

        if (__inline_fn)
                return;

        if (stmt->type == STMT_COMPOUND)
                add_ptr_list(&stmt_list, stmt);

        if (!is_outer_stmt(stmt))
                return;

        FOR_EACH_PTR(stmt_list, tmp) {
                match_end_of_block(tmp);
        } END_FOR_EACH_PTR(tmp);
        free_ptr_list(&stmt_list);
}

static void match_end_func(struct symbol *sym)
{
        if (__inline_fn)
                return;
        free_ptr_list(&stmt_list);
}

void register_scope(int id)
{
        add_hook(&match_stmt, STMT_HOOK_AFTER);
        add_hook(&match_end_func, AFTER_FUNC_HOOK);
}
```

```
**********************************************************
     738 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/add_gfp_to_allocations.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

if ! test -e kernel.allocation_funcs || ! test -e kernel.gfp_flags ; then
    echo "We need the kernel.allocation_funcs and the kernel.gfp_flags files"
    echo "The scripts to generate them are in smatch_data/"
    exit 1
fi

bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/kernel.allocation_funcs_gfp.remove)
tmp=$(mktemp /tmp/smatch.XXXX)

echo "// Automatically generated by add_gfp_to_allocations.sh" > kernel.allocati
for i in $(grep -v "//" kernel.allocation_funcs) ; do
    if ! grep -w $i kernel.gfp_flags ; then
        echo $i X
    fi
done >> $tmp

cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> kernel.allocation_func
rm $tmp

echo "Done.  Created kernel.allocation_funcs_gfp"
```

```
**********************************************************
    1813 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/build_generic_data.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

# This is a generic script to parse --info output.  For the kernel, don't use
# this script, use build_kernel_data.sh instead.

NR_CPU=$(cat /proc/cpuinfo | grep ^processor | wc -l)
SCRIPT_DIR=$(dirname $0)
DATA_DIR=smatch_data
PROJECT=smatch_generic
TARGET=""

function usage {
    echo
    echo "Usage:  $0"
    echo "Updates the smatch_data/ directory and builds the smatch database"
    echo " -p <project> (default = $PROJECT)"
    echo
    exit 1
}

while true ; do
    if [[ "$1" == "--target" ]] ; then
        shift
        TARGET="$1"
        shift
    elif [ "$1" == "-p" ] || [ "$1" == "--project" ] ; then
        shift
        PROJECT="$1"
        shift
    elif [ "$1" == "--help" ] || [ "$1" = "-h" ] ; then
        usage
    else
        break
    fi
done

if [ -e $SCRIPT_DIR/../smatch ] ; then
    BIN_DIR=$SCRIPT_DIR/../
else
    echo "This script should be located in the smatch_scripts/ subdirectory of t
    exit 1
fi

# If someone is building the database for the first time then make sure all the
# required packages are installed
if [ ! -e smatch_db.sqlite ] ; then
    [ -e smatch_warns.txt ] || touch smatch_warns.txt
    if ! $SCRIPT_DIR/../smatch_data/db/create_db.sh -p=$PROJECT smatch_warns.txt
        echo "Hm... Not working.  Make sure you have all the sqlite3 packages"
        echo "And the sqlite3 libraries for Perl and Python"
        exit 1
    fi
fi

make -j${NR_CPU} CHECK="$BIN_DIR/smatch --call-tree --info --param-mapper --spam

find -name \*.c.smatch -exec cat \{\} \; -exec rm \{\} \; > smatch_warns.txt

for i in $SCRIPT_DIR/gen_* ; do
        $i smatch_warns.txt -p=${PROJECT}
```



```
done

mkdir -p $DATA_DIR
mv $PROJECT.* $DATA_DIR

$SCRIPT_DIR/../smatch_data/db/create_db.sh -p=$PROJECT smatch_warns.txt
```

```
**********************************************************
    1257 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/build_kernel_data.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

PROJECT=kernel

function usage {
    echo
    echo "Usage:  $0"
    echo "Updates the smatch_data/ directory and builds the smatch database"
    echo
    exit 1
}

if [ "$1" = "-h" ] || [ "$1" = "--help" ] ; then
        usage;
fi

SCRIPT_DIR=$(dirname $0)
if [ -e $SCRIPT_DIR/../smatch -a -d kernel -a -d fs ] ; then
    CMD=$SCRIPT_DIR/../smatch
    DATA_DIR=$SCRIPT_DIR/../smatch_data
else
    echo "This script should be located in the smatch_scripts/ subdirectory of t
    echo "It should be run from the root of a kernel source tree."
    exit 1
fi

# If someone is building the database for the first time then make sure all the
# required packages are installed
if [ ! -e smatch_db.sqlite ] ; then
    [ -e smatch_warns.txt ] || touch smatch_warns.txt
    if ! $DATA_DIR/db/create_db.sh -p=kernel smatch_warns.txt ; then
        echo "Hm... Not working.  Make sure you have all the sqlite3 packages"
        echo "And the sqlite3 libraries for Perl and Python"
        exit 1
    fi
fi

$SCRIPT_DIR/test_kernel.sh --call-tree --info --param-mapper --spammy --data=$DA

for i in $SCRIPT_DIR/gen_* ; do
        $i smatch_warns.txt -p=kernel
done

mv ${PROJECT}.* $DATA_DIR

$DATA_DIR/db/create_db.sh -p=kernel smatch_warns.txt
```

```
**********************************************************
    2569 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/call_tree.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl

# This script is supposed to help use the param_mapper output.
# Give it a function and parameter and it lists the functions
# and parameters which are basically equivalent.

use strict;

sub usage()
{
    print("call_tree.pl <smatch output file>\n");
    print("call_tree.pl finds paths between two functions\n");
    exit(1);
}

my %param_map;

my $UNKNOWN  = 1;
my $NOTFOUND = 2;
my $FOUND    = 3;

my $path;

sub print_path()
{
    my $i = 0;

    foreach my $func (@{$path}) {
        if ($i++) {
            print(", ");
        }
        print("$func");
    }
    print("\n");
    print("\n");
}

sub recurse($$)
{
    my $link = shift;
    my $target = shift;
    my $found = 0;

    if ($link =~ /$target/) {
        print_path();
        return 1;
    }
    if (%{$param_map{$link}}->{found} == $NOTFOUND) {
        return 0;
    }

    %{$param_map{$link}}->{found} = $NOTFOUND;

    foreach my $l (@{%{$param_map{$link}}->{links}}){
        push(@{$path}, $l);
        $found = recurse($l, $target);
        if (!$found) {
            pop(@{$path});
        } else {
            last;
        }
    }

    return $found;
}

sub search($$)
{
    my $start_func = shift;
    my $end_func = shift;

    foreach my $link (@{%{$param_map{$start_func}}->{links}}){
        %{$param_map{$start_func}}->{found} = $NOTFOUND;
        foreach my $l (@{%{$param_map{$start_func}}->{links}}){
            %{$param_map{$l}}->{found} = $NOTFOUND;
        }
        $path = [$start_func, $link];
        %{$param_map{$link}}->{found} = $UNKNOWN;
        recurse($link, $end_func);
    }
}

sub add_link($$)
{
    my $one = shift;
    my $two = shift;

    if (!defined($param_map{$one})) {
        $param_map{$one} = {found => $UNKNOWN, links => []};
    }
    push @{$param_map{$one}->{links}}, $two;
}

sub load_all($)
{
    my $file = shift;

    open(FILE, "<$file");
    while (<FILE>) {
        if (/.*?:\d+ (.*?)\(\) info: func_call (.*)/) {
            add_link("$1", "$2");
        }
    }
}

sub set_all_unknown()
{
    my $i = 0;

    foreach my $func (keys %param_map){
        %{$param_map{$func}}->{found} = $UNKNOWN;
    }
}

my $file = shift();
if (!$file) {
    usage();
}

if (! -e $file) {
    printf("Error:  $file does not exist.\n");
    exit(1);
}

print("Loading functions...\n");
load_all($file);
```

```
128 while (1) {
129     my $start_func;
130     my $end_func;

132     print("Enter the start function:  ");
133     $start_func = <STDIN>;
134     $start_func =~ s/^\s+|\s+$//g;
135     print("Enter the target function:  ");
136     $end_func = <STDIN>;
137     $end_func =~ s/^\s+|\s+$//g;


140     print("$start_func to $end_func\n");
141     if ($start_func =~ /./ && $end_func =~ /./) {
142         search($start_func, $end_func);
143     }

145     set_all_unknown();
146 }
```

```
**********************************************************
    2575 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/filter_kernel_deref_check.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages>"
    exit 1
fi

IFS="
"

for line in $(grep 'dereferenced before' $file) ; do

    code_file=$(echo "$line" | cut -d ':' -f1)
    lineno=$(echo "$line" | cut -d ' ' -f1 | cut -d ':' -f2)
    function=$(echo "$line" | cut -d ' ' -f2)
    variable=$(echo "$line" | cut -d "'" -f3)
    source_line=$(tail -n +$lineno $code_file | head -n 1 | sed -e 's/^\W*//')

    if echo "$source_line" | grep -q rcu_assign_pointer ; then
        continue
    fi
    if echo "$source_line" | grep -q '^\W*tda_' ; then
        continue
    fi
    if echo "$source_line" | grep -q tda_fail ; then
        continue
    fi
    if echo "$source_line" | grep -q '^\W*ATH5K_' ; then
        continue
    fi
    if echo "$source_line" | grep -qw CMDINFO ; then
        continue
    fi
    if echo "$source_line" | grep -qw dump_desc_dbg ; then
        continue
    fi
    if echo "$source_line" | grep -qw CAMERA_IS_OPERATIONAL ; then
        continue
    fi
    if echo "$source_line" | grep -qw USBVISION_IS_OPERATIONAL ; then
        continue
    fi
    if echo "$source_line" | grep -qw DEV_INIT_TEST_WITH_RETURN ; then
        continue
    fi
    if echo "$source_line" | grep -qw TW_PRINTK ; then
        continue
    fi
    if echo "$source_line" | grep -qw RESET_ONE_SEC_TX_CNT ; then
        continue
    fi
    if echo "$source_line" | grep -qw SOCK_DEBUG; then
        continue
    fi
    if echo "$source_line" | grep -qw P80211SKB_RXMETA ; then
        continue
    fi
    if echo "$source_line" | grep -qw ACM_READY ; then
        continue
```



```
    fi
    if echo "$source_line" | grep -qw v4l2_subdev_notify ; then
        continue
    fi
    if echo "$source_line" | egrep -qw 'tuner_(err|info)' ; then
        continue
    fi
    if echo "$source_line" | grep -qw DBG_SKB ; then
        continue
    fi
    if echo "$source_line" | grep -qw for_each_mddev ; then
        continue
    fi
    if echo "$source_line" | grep -qw v4l2_subdev_call ; then
        continue
    fi
    if echo "$source_line" | grep -qw VALID_CALLBACK ; then
        continue
    fi
    if [ "$variable" == "bp->dev" ] && echo "$source_line" | grep -qw DP ; then
        continue
    fi
    if echo "$source_line" | grep -qw BNX2X_ERR ; then
        continue
    fi
    if echo "$source_line" | grep -qw FCOE_NETDEV_DBG ; then
        continue
    fi
    if echo "$source_line" | grep -qw __rq_for_each_bio ; then
        continue
    fi
    if echo "$source_line" | grep -qw IPS_DMA_DIR ; then
        continue
    fi
    if [ "$variable" == "dev" ] && echo "$source_line" | grep -qw dprintk ; then
        continue
    fi

    echo "$code_file:$lineno $function '$variable': $source_line"
done
```

```
**********************************************************
     254 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/find_expanded_holes.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl

use strict;

my $cur_struct = "";
my $printed = 0;

while (<>) {
    if ($_ =~ /^struct (\w+) {/) {
        $cur_struct = $1;
        $printed = 0;
        next;
    }
    if ($_ =~ /.* hole,.*/ && !$printed) {
        print "$cur_struct\n";
        $printed = 1;
    }
}
```

```
**********************************************************
     407 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/find_null_params.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages>"
    exit 1
fi

grep " unchecked " $file | cut -d ' ' -f 5- | sort -u > unchecked
grep " undefined " $file | cut -d ' ' -f 5- | sort -u > null_calls.txt
cat null_calls.txt unchecked | sort | uniq -d > null_params.txt
IFS="
"
for i in $(cat null_params.txt) ; do
        grep "$i" $file | grep -w undefined
done
```

```
**********************************************************
    1653 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/follow_params.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl

use strict;

sub usage()
{
    print "$0 <smatch output file> <function> <parameter>\n";
    print "Give this program a function and parameter and it follows to find\n";
    print "how the parameter gets passed down to lower levels.\n'";
    exit(1);
}

my %param_map;

my $UNUSED   = 0;
my $USED     = 1;

sub print_link($)
{
    my $link = shift;

    $link =~ s/%/ /;
    print "$link\n";
}

sub recurse($)
{
    my $link = shift;

    if ($param_map{$link}{used} == $USED) {
        return;
    }
    ${param_map}{$link}->{used} = $USED;

    print_link($link);

    foreach my $l (@{$param_map{$link}{links}}){
        recurse($l);
    }

}

sub follow($$)
{
    my $f = shift;
    my $p = shift;

    recurse("$f%$p");
}

sub add_link($$)
{
    my $one = shift;
    my $two = shift;

    if (!defined($param_map{$one})) {
        $param_map{$one} = {used => $UNUSED, links => []};
    }
    push @{$param_map{$one}{links}}, $two;
}
```



```
sub load_all($)
{
    my $file = shift;

    open(FILE, "<$file");
    while (<FILE>) {
        if (/.*?:\d+ (.*?)\(\) info: param_mapper (\d+) => (.*?) (\d+)/) {
            add_link("$1%$2", "$3%$4");
        }
    }
}

sub set_all_unused()
{
    foreach my $func (keys %param_map){
        ($param_map{$func}{used} = $UNUSED);
    }

}

my $file = shift();
my $func = shift();
my $param = shift();

if (!defined($file) or !defined($func) or !defined($param)) {
    usage();
}

if (! -e $file) {
    printf("Error:  $file does not exist.\n");
    exit(1);
}

load_all($file);

while (1) {
    follow($func, $param);

    $func = shift();
    $param = shift();
    if (!defined($func) || !defined($param)) {
        last;
    }
    set_all_unused();
}
```

```
**********************************************************
     850 Fri Dec 21 15:00:29 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_allocation_list.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/kernel.allocation_funcs.remove)
tmp=$(mktemp /tmp/smatch.XXXX)

echo "// list of functions that return a new allocation." \
    > kernel.allocation_funcs
echo '// generated by "gen_allocation_list.sh"' >> kernel.allocation_funcs
grep "allocation func$" $file | cut -s -d ' ' -f 2 | cut -d '(' -f 1 | \
    sort -u > $tmp
echo "kmalloc" >> $tmp
echo "kzalloc" >> $tmp
echo "kcalloc" >> $tmp
echo "__alloc_skb" >> $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u \
    >> kernel.allocation_funcs
rm $tmp
echo "Done.  List saved as 'kernel.allocation_funcs'"
```

**********
673 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_bit_shifters.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 #!/bin/bash

 3 file=$1
 4 project=$(echo "$2" | cut -d = -f 2)

 6 if [[ "$file" = "" ]] ; then
 7     echo "Usage:  $0 <file with smatch messages> -p=<project>"
 8     exit 1
 9 fi

11 bin_dir=$(dirname $0)
12 remove=$(echo ${bin_dir}/../smatch_data/${project}.bit_shifters.remove)
13 tmp=$(mktemp /tmp/smatch.XXXX)

15 echo "// list of macros used as shifters." \
16     > ${project}.bit_shifters
17 echo '// generated by `gen_bit_shifters.sh'' >> ${project}.bit_shifters
18 grep "info: bit shifter" $file | cut -s -d "'" -f 2- | sed -e "s/'//g" | sort -u

20 cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> ${project}.bit_shifter
21 rm $tmp
22 echo "Done.  List saved as '${project}.bit_shifters'"
```

```
**********************************************************
     786 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_dma_funcs.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

outfile="kernel.dma_funcs"
bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/${outfile}.remove)
tmp=$(mktemp /tmp/smatch.XXXX)
tmp2=$(mktemp /tmp/smatch.XXXX)

echo "// list of DMA function and buffer parameters." > $outfile
echo '// generated by 'gen_dma_funcs.sh'' >> $outfile
${bin_dir}/trace_params.pl $file usb_control_msg 6 >> $tmp
${bin_dir}/trace_params.pl $file usb_fill_bulk_urb 3 >> $tmp
cat $tmp | sort -u > $tmp2
mv $tmp2 $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> $outfile
rm $tmp
echo "Done.  List saved as '$outfile'"
```

**********
*746 Fri Dec 21 15:00:30 2018*
*new/usr/src/tools/smatch/src/smatch_scripts/gen_err_ptr_list.sh*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
 1 #!/bin/bash

 3 file=$1
 4 project=$(echo "$2" | cut -d = -f 2)

 6 if [[ "$file" = "" ]] ; then
 7     echo "Usage:  $0 <file with smatch messages> -p=<project>"
 8     exit 1
 9 fi

11 if [[ "$project" != "kernel" ]] ; then
12     exit 0
13 fi

15 bin_dir=$(dirname $0)
16 remove=$(echo ${bin_dir}/../smatch_data/kernel.returns_err_ptr.remove)
17 tmp=$(mktemp /tmp/smatch.XXXX)

19 echo "// list of functions that return a new allocation." \
20     > kernel.returns_err_ptr
21 echo '// generated by `gen_err_ptr_list.sh`' >> kernel.returns_err_ptr
22 grep "returns_err_ptr$" $file | cut -s -d ' ' -f 2 | cut -d '(' -f 1 | \
23     sort -u > $tmp
24 cat $tmp $remove $remove 2> /dev/null | sort | uniq -u \
25     >> kernel.returns_err_ptr
26 rm $tmp
27 echo "Done.  List saved as 'kernel.returns_err_ptr'"
```

```
**********************************************************
     710 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_expects_err_ptr.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

outfile="kernel.expects_err_ptr"
bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/${outfile}.remove)
tmp=$(mktemp /tmp/smatch.XXXX)

echo "// list of functions which expect an ERR_PTR." > $outfile
echo '// generated by `gen_expects_err_ptr.sh`' >> $outfile
grep -w "expects ERR_PTR" $file | cut -d ' ' -f 2,6 | \
    sed -e 's/([1234567890]*)//' | sort -u > $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> $outfile
rm $tmp
echo "Done.  List saved as '$outfile'"
```

```
**********************************************************
     683 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_frees_list.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/kernel.frees_argument.remove)
tmp=$(mktemp /tmp/smatch.XXXX)

echo "// list of functions and the argument they free." > kernel.frees_argument
echo '// generated by `gen_frees_list.sh'' >> kernel.frees_argument
grep -w free_arg $file | cut -d ' ' -f 5- >> $tmp
cat $tmp $remove 2> /dev/null | sort | uniq -u >> kernel.frees_argument
rm $tmp
echo "Done.  List saved as 'kernel.frees_argument'"
```

```
**********************************************************
     804 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_gfp_flags.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

outfile="kernel.gfp_flags"
bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/${outfile}.remove)
tmp=$(mktemp /tmp/smatch.XXXX)
tmp2=$(mktemp /tmp/smatch.XXXX)

echo "// list of GFP flag parameters." > $outfile
echo '// generated by `gen_gfp_flags.sh'' >> $outfile
${bin_dir}/trace_params.pl $file kmalloc 1 >> $tmp
${bin_dir}/trace_params.pl $file kzalloc 1 >> $tmp
${bin_dir}/trace_params.pl $file kcalloc 2 >> $tmp
cat $tmp | sort -u > $tmp2
mv $tmp2 $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> $outfile
rm $tmp
echo "Done.  List saved as '$outfile'"
```

**********
*795 Fri Dec 21 15:00:30 2018*
*new/usr/src/tools/smatch/src/smatch_scripts/gen_implicit_dependencies.sh*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/kernel.implicit_dependencies.remove)
tmp=$(mktemp /tmp/smatch.XXXX)

# echo "// list of syscalls and the fields they write/read to." > kernel.implici
# echo '// generated by 'gen_implicit_dependencies.sh'' >> kernel.implicit_depen
grep -w read_list $file >> $tmp
grep -w write_list $file >> $tmp
# cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> kernel.implicit_depe
cat $tmp >> kernel.implicit_dependencies
rm $tmp
echo "Done.  List saved as 'kernel.implicit_dependencies"
```

```
**********************************************************
     913 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_no_return_funcs.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

outfile="${project}.no_return_funcs"
bin_dir=$(dirname $0)
add_file=$(echo ${bin_dir}/../smatch_data/${outfile}.add)
remove=$(echo ${bin_dir}/../smatch_data/${outfile}.remove)
tmp=$(mktemp /tmp/smatch.XXXX)
tmp2=$(mktemp /tmp/smatch.XXXX)

echo "// list of functions which don't return." > $outfile
echo '// generated by `gen_no_return_funcs.sh'' >> $outfile
cat $(echo ${bin_dir}/../smatch_data/no_return_funcs) >> $outfile
cat $add_file >> $outfile 2> /dev/null

grep no_return_funcs $file | cut -d ' ' -f 2 | cut -d '(' -f 1 > $tmp

cat $tmp | sort -u > $tmp2
mv $tmp2 $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> $outfile
rm $tmp
echo "Done.  List saved as '$outfile'"
echo "Copy it to smatch_data/<project>.no_return_funcs"
```

```
**********************************************************
     693 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_puts_list.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/kernel.puts_argument.remove)
tmp=$(mktemp /tmp/smatch.XXXX)

echo "// list of functions and the argument they decrement the ref of." > kernel
echo '// generated by 'gen_puts_list.sh'' >> kernel.puts_argument
grep -w puts_arg $file | cut -d ' ' -f 5- >> $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> kernel.puts_argument
rm $tmp
echo "Done.  List saved as 'kernel.puts_argument'"
```

```
**********************************************************
     756 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_returns_held.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/kernel.returns_held.remove)
tmp=$(mktemp /tmp/smatch.XXXX)

echo "// list of functions that return a held device." \
    > kernel.returns_held_funcs
echo '// generated by `gen_returns_held.sh`' >> kernel.returns_held_funcs
grep "returned dev is held" $file | cut -s -d ' ' -f 2 | cut -d '(' -f 1 | \
    sort -u > $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u \
    >> kernel.returns_held_funcs
rm $tmp
echo "Done.  List saved as 'kernel.returns_held_funcs'"
```

```
**********************************************************
     794 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_rosenberg_funcs.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

outfile="kernel.rosenberg_funcs"
bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/${outfile}.remove)
tmp=$(mktemp /tmp/smatch.XXXX)
tmp2=$(mktemp /tmp/smatch.XXXX)

echo "// list of copy_to_user function and buffer parameters." > $outfile
echo '// generated by 'gen_rosenberg_funcs.sh'' >> $outfile
${bin_dir}/trace_params.pl $file copy_to_user 1 >> $tmp
${bin_dir}/trace_params.pl $file nla_put 3 >> $tmp
cat $tmp | sort -u > $tmp2
mv $tmp2 $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> $outfile
rm $tmp
echo "Done.  List saved as '$outfile'"
```

```
**********************************************************
     852 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_sizeof_param.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

outfile="${project}.sizeof_param"
bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/${outfile}.remove)
tmp=$(mktemp /tmp/smatch.XXXX)
tmp2=$(mktemp /tmp/smatch.XXXX)


echo "// list of function parameters that are the size of a buffer." > $outfile
echo '// generated by 'gen_sizeof_param.sh'' >> $outfile

grep sizeof_param $file | grep '[0-9] [0-9]$' | cut -d ' ' -f 5- | \
    sort -u | sed -e "s/'//g" > $tmp
grep sizeof_param $file | grep '[0-9] -1$' | cut -d ' ' -f 5- | \
    sort -u | sed -e "s/'//g" >> $tmp
grep -f $remove $tmp >> $tmp2 2> /dev/null
cat $tmp $tmp2 2> /dev/null | sort | uniq -u >> $outfile
rm $tmp
rm $tmp2

echo "Done.  List saved as '$outfile'"
```

```
**********************************************************
     629 Fri Dec 21 15:00:30 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_trinity.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

cat << EOF > trinity_smatch.h

#pragma once

/* Syscalls from arch/x86/syscalls/syscall_64.tbl */

#include "sanitise.h"
#include "syscall.h"
#include "syscalls/syscalls.h"

EOF

cat smatch_trinity_* >> trinity_smatch.c


for i in $(grep syscallentry smatch_trinity_*  | cut -d ' ' -f 3) ; do
    echo "extern struct syscallentry $i;" >> trinity_smatch.h
done

echo "" >> trinity_smatch.h
echo "struct syscalltable syscalls_smatch[] = {" >> trinity_smatch.h

for i in $(grep syscallentry smatch_trinity_*  | cut -d ' ' -f 3) ; do
    echo "{ .entry = &$i }," >> trinity_smatch.h
done

echo "};" >> trinity_smatch.h
```

```
**********************************************************
     738 Fri Dec 21 15:00:31 2018
new/usr/src/tools/smatch/src/smatch_scripts/gen_unwind_functions.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

file=$1
project=$(echo "$2" | cut -d = -f 2)

if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 <file with smatch messages> -p=<project>"
    exit 1
fi

if [[ "$project" != "kernel" ]] ; then
    exit 0
fi

outfile="kernel.unwind_functions"
bin_dir=$(dirname $0)
remove=$(echo ${bin_dir}/../smatch_data/${outfile}.remove)
tmp=$(mktemp /tmp/smatch.XXXX)
tmp2=$(mktemp /tmp/smatch.XXXX)

echo "// list of unwind functions." > $outfile
echo '// generated by 'gen_unwind_functions.sh'' >> $outfile
grep "is unwind function" $file | cut -d ' ' -f 2 | cut -d '(' -f 1 >> $tmp
cat $tmp | sort -u > $tmp2
mv $tmp2 $tmp
cat $tmp $remove $remove 2> /dev/null | sort | uniq -u >> $outfile
rm $tmp
echo "Done.  List saved as '$outfile'"
```

```
**********************************************************
    1285 Fri Dec 21 15:00:31 2018
new/usr/src/tools/smatch/src/smatch_scripts/generisize.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl

use strict;

sub help()
{
    print "usage: $0 [-r]\n";
    print "Counts the number of errors of each type.\n";
    print "-r means down to the nearest 10.\n";
    exit 1;
}

my $round;
my $arg = shift;
if ($arg =~ /-h/) {
    help();
} elsif ($arg =~ /-r/) {
    $round = 1;
}

my %msgs;

sub add_msg($)
{
    my $msg = shift;

    if (defined $msgs{$msg}) {
        $msgs{$msg}++;
    } else {
        $msgs{$msg} = 1;
    }
}

while (<>) {
    s/^.*?:\d+(|:\d+:) .*? //;
    s/[us](16|32|64)(min|max)//g;
    s/0x\w+//g;
    s/[01234567890]//g;
    if ($_ =~ /can't/) {
        s/(.*can't.*').*?('.*)/$1 $2/;
        s/(.*?)'.*?'(.*can't.*)/$1 $2/;
    } elsif ($_ =~ /don't/) {
        s/(.*don't.*').*?('.*)/$1 $2/;
    } else {
        s/'.*?'/''/g;
    }
    s/,//g;
    s/\(\w+ returns null\)/(... returns null)/;
    s/dma on the stack \(.*?\)/dma on the stack (,..)/;
    s/possible ERR_PTR '' to .*/possible ERR_PTR '' to .../;
    s/inconsistent returns ([^ ]+?) locked \(\)/inconsistent returns ... locked
    s/(.*) [^ ]* (too large for) [^ ]+ (.*)/$1 $2 $3/;

    add_msg($_);
}

foreach my $key (sort { $msgs{$b} <=> $msgs{$a} } keys %msgs) {
    my $count = $msgs{$key};

    if ($round) {
```



```
        $count = $msgs{$key} - $msgs{$key} % 10;
    }
    print "$count $key";
}
```

```
**********************************************************
     237 Fri Dec 21 15:00:31 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/README
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 Python module for parsing kernel.implicit_dependencies

   3 'python main.py -h' for usage.
   4 run 'python main.py -v -p' to generate verbose and pretty print.

   6 To generate kernel.implicit_dependencies, run 'smatch_scripts/build_kernel_data.
```

**********
696 Fri Dec 21 15:00:31 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/constants.py
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
from collections import defaultdict

# location of kernel.implicit_dependencies
IMPL_DEP_FILE_STR = "../../smatch_data/kernel.implicit_dependencies"
OUTPUT_FILE_STR = "implicit_dependencies"

# struct fields to ignore, because they are too common
GLOBAL_BLACKLIST = [
    ('fd', 'file'),
]

# here we can manually add struct fields that smatch missed
hardcode_syscall_write_fields = {}

# here we can manually add struct fields that smatch missed
hardcode_syscall_read_fields = {
    "msync": [("vm_area_struct", "vm_flags"), ("vm_area_struct", "vm_file")]
}

SYSCALL_PREFIXES = [
    "SYSC_",
    "C_SYSC_",
    "sys_",
]

class ListType(object):
    READ = "read_list"
    WRITE = "write_list"
```

```
**********************************************************
    1051 Fri Dec 21 15:00:31 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/main.py
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 import argparse
 2 import sys

 4 from constants import (
 5     IMPL_DEP_FILE_STR,
 6     OUTPUT_FILE_STR,
 7 )
 8 from parser import Parser

10 def main():
11     arg_parser = argparse.ArgumentParser(
12         description="Control module for tracking implicit dependencies"
13     )
14     arg_parser.add_argument(
15         "-f", "--file", default=IMPL_DEP_FILE_STR,
16         help="path to kernel.implicit_dependencies",
17     )
18     arg_parser.add_argument(
19         "-o", "--output", default=OUTPUT_FILE_STR,
20         help="where to output info",
21     )
22     arg_parser.add_argument(
23         "-v", "--verbose", action="store_true",
24         help="if verbose, we list what fields are responsible for the dependency
25     )
26     arg_parser.add_argument(
27         "-p", "--pretty", action="store_true",
28         help="print implicit dependencies in pretty format"
29     )
30     args = arg_parser.parse_args()

32     p = Parser(args.file, output_file_str=args.output, verbose=args.verbose, pre
33     p.parse()
34     p.write()
35     p.close()


38 if __name__ == "__main__":
39     sys.exit(main())
```

```
**********************************************************
    6030 Fri Dec 21 15:00:31 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/parser.py
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 from collections import defaultdict
 2 import copy
 3 import json
 4 import sys
 5 import pprint

 7 from constants import (
 8     GLOBAL_BLACKLIST,
 9     IMPL_DEP_FILE_STR,
10     OUTPUT_FILE_STR,
11     SYSCALL_PREFIXES,
12     ListType,
13     hardcode_syscall_read_fields,
14     hardcode_syscall_write_fields,
15 )

17 class Parser(object):
18     def __init__(
19         self,
20         impl_dep_file_str=IMPL_DEP_FILE_STR,
21         output_file_str=OUTPUT_FILE_STR,
22         verbose=False,
23         pretty=False
24     ):
25         try:
26             self.impl_dep_file = file(impl_dep_file_str, 'r')
27             self.output_file = file(output_file_str + '.json', 'w+')
28             if verbose:
29                 self.output_file_verbose = file(output_file_str + '_verbose.json
30             if pretty:
31                 self.pretty_output_file = file(output_file_str + '.pretty', 'w+'
32                 self.pretty_output_file_verbose = file(output_file_str + '_verbo
33         except IOError:
34             sys.stderr.write("ERROR: Cannot open files %s %s.\n" % (impl_dep_fil
35             sys.exit(1)
36         self.verbose = verbose
37         self.pretty = pretty
38         self.syscall_read_fields = defaultdict(set)
39         self.syscall_write_fields = defaultdict(set)
40         self.implicit_dependencies = defaultdict(set)
41         self.verbose_impl_dep = defaultdict(list)
42         self.deref_counter = defaultdict(int)  # count which struct->members are

44         for syscall,fields in hardcode_syscall_read_fields.iteritems():
45             self.syscall_read_fields[syscall].update(set(fields))

47         for syscall,fields in hardcode_syscall_write_fields.iteritems():
48             self.syscall_write_fields[syscall].update(set(fields))

50     def _sanitize_syscall(self, syscall):
51         for prefix in SYSCALL_PREFIXES:
52             if syscall.startswith(prefix):
53                 return syscall[len(prefix):]
54         return syscall

56     def _deref_to_tuple(self, deref):
57         """ (struct a)->b ==> (a,b) """
58         struct, member = deref.split('->')
59         struct = struct[1:-1]  # strip parens
60         struct = struct.split(' ')[1]  # drop struct keyword
```



```
61         return (struct, member)

63     def _split_field(self, field):
64         field = field.strip()
65         field = field[1: -1]  # strip square brackets
66         derefs = [struct.strip() for struct in field.strip().split(',') if struc
67         return map(
68             lambda deref: self._deref_to_tuple(deref),
69             derefs
70         )

72     def _sanitize_line(self, line):
73         syscall_and_listtype, field = line.split(':')
74         syscall, list_type = syscall_and_listtype.split(' ')
75         syscall = self._sanitize_syscall(syscall)
76         derefs = self._split_field(field)
77         return syscall, list_type, derefs

79     def _add_fields(self, syscall, list_type, derefs):
80         if list_type == ListType.READ:
81             d = self.syscall_read_fields
82         elif list_type == ListType.WRITE:
83             d = self.syscall_write_fields
84         for deref in derefs:
85             if deref in GLOBAL_BLACKLIST:  # ignore spammy structs
86                 continue
87             d[syscall].add(deref)

89     def _construct_implicit_deps(self):
90         """ just do a naive O(n^2) loop to see intersections between write_list
91         for this_call,read_fields in self.syscall_read_fields.iteritems():
92             for that_call,write_fields in self.syscall_write_fields.iteritems():
93                 if that_call == this_call:  # calls are obviously dependent on t
94                     continue
95                 intersection = read_fields & write_fields
96                 if intersection:
97                     self.implicit_dependencies[this_call].add(that_call)
98                 if intersection and self.verbose:
99                     self.verbose_impl_dep[this_call].append({
100                        'call': that_call,
101                        'reason': intersection,
102                    })
103                    for deref in intersection:
104                        self.deref_counter[deref] += 1

106    def parse(self):
107        for line in self.impl_dep_file:
108            syscall, list_type, derefs = self._sanitize_line(line)
109            self._add_fields(syscall, list_type, derefs)
110        # pprint.pprint(dict(self.syscall_write_fields))
111        # pprint.pprint(dict(self.syscall_read_fields))
112        self._construct_implicit_deps()
113        # pprint.pprint(dict(self.implicit_dependencies))
114        # pprint.pprint(dict(self.verbose_impl_dep))

116    def _listify_verbose_reason(self, reason):
117        r = copy.deepcopy(reason)
118        r['reason'] = list(r['reason'])
119        r['reason'] = map(
120            lambda (struct,field): struct + '->' + field,
121            r['reason']
122        )
123        return r

125    def _get_json_dependencies(self):
126        implicit_dependencies = {}
```

```
127         verbose_impl_dep = {}
128         for call, dep_set in self.implicit_dependencies.iteritems():
129             implicit_dependencies[call] = list(dep_set)
130         for call, call_reasons in self.verbose_impl_dep.iteritems():
131             verbose_impl_dep[call] = map(
132                 lambda reason: self._listify_verbose_reason(reason),
133                 call_reasons,
134             )
135         return implicit_dependencies, verbose_impl_dep

137     def write(self):
138         implicit_dependencies, verbose_impl_dep = self._get_json_dependencies()
139         json.dump(implicit_dependencies, self.output_file)
140         if self.verbose:
141             json.dump(verbose_impl_dep, self.output_file_verbose)
142         if self.pretty:
143             pprint.pprint(dict(self.implicit_dependencies), self.pretty_output_f
144             pprint.pprint(dict(self.verbose_impl_dep), self.pretty_output_file_v
145         for deref, count in sorted(self.deref_counter.iteritems(), key=lambda (k
146             print "%s: %d" % (deref, count)

148     def close(self):
149         self.output_file.close()
150         self.impl_dep_file.close()
151         if self.verbose:
152             self.output_file_verbose.close()
```

```
**********************************************************
  387783 Fri Dec 21 15:00:31 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/with_structs/i
mplicit_dependencies
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
{'accept4': set(['accept4',
                 'bind',
                 'bpf',
                 'connect',
                 'copy_file_range',
                 'epoll_ctl',
                 'epoll_wait',
                 'fallocate',
                 'fchdir',
                 'fchmod',
                 'fchown',
                 'fcntl',
                 'fcntl64',
                 'fdatasync',
                 'fgetxattr',
                 'flistxattr',
                 'flock',
                 'fremovexattr',
                 'fsetxattr',
                 'fstatfs',
                 'fstatfs64',
                 'fsync',
                 'ftruncate',
                 'futimesat',
                 'getdents',
                 'getdents64',
                 'getpeername',
                 'getsockname',
                 'getsockopt',
                 'inotify_add_watch',
                 'inotify_rm_watch',
                 'ioctl',
                 'listen',
                 'llseek',
                 'lseek',
                 'mq_getsetattr',
                 'mq_notify',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'old_readdir',
                 'perf_event_open',
                 'pread64',
                 'preadv',
                 'preadv2',
                 'preadv64',
                 'preadv64v2',
                 'pwrite64',
                 'pwritev',
                 'pwritev2',
                 'pwritev64',
                 'pwritev64v2',
                 'read',
                 'readahead',
                 'readv',
                 'recvfrom',
                 'sendfile',
                 'sendfile64',
                 'sendto',
                 'setsockopt',
```



```
                 'shutdown',
                 'signalfd4',
                 'splice',
                 'sync_file_range',
                 'syncfs',
                 'tee',
                 'utime',
                 'utimensat',
                 'vmsplice',
                 'write',
                 'writev']),
 'acct': set(['accept4',
              'acct',
              'dup',
              'dup3',
              'epoll_create1',
              'epoll_ctl',
              'eventfd2',
              'flock',
              'memfd_create',
              'mmap_pgoff',
              'mq_open',
              'open',
              'open_by_handle_at',
              'openat',
              'perf_event_open',
              'pipe2',
              'remap_file_pages',
              'setns',
              'shmat',
              'shmctl',
              'shmdt',
              'socket',
              'socketpair',
              'swapoff',
              'swapon',
              'uselib']),
 'alarm': set(['adjtimex',
               'alarm',
               'clock_adjtime',
               'getitimer',
               'getrusage',
               'ppoll',
               'select',
               'setitimer',
               'settimeofday',
               'wait4',
               'waitid']),
 'bind': set(['accept4',
              'bind',
              'bpf',
              'connect',
              'copy_file_range',
              'epoll_ctl',
              'epoll_wait',
              'fallocate',
              'fchdir',
              'fchmod',
              'fchown',
              'fcntl',
              'fcntl64',
              'fdatasync',
              'fgetxattr',
              'flistxattr',
              'flock',
              'fremovexattr',
```

```
126             'fsetxattr',
127             'fstatfs',
128             'fstatfs64',
129             'fsync',
130             'ftruncate',
131             'futimesat',
132             'getdents',
133             'getdents64',
134             'getpeername',
135             'getsockname',
136             'getsockopt',
137             'inotify_add_watch',
138             'inotify_rm_watch',
139             'ioctl',
140             'listen',
141             'llseek',
142             'lseek',
143             'mq_getsetattr',
144             'mq_notify',
145             'mq_timedreceive',
146             'mq_timedsend',
147             'old_readdir',
148             'perf_event_open',
149             'pread64',
150             'preadv',
151             'preadv2',
152             'preadv64',
153             'preadv64v2',
154             'pwrite64',
155             'pwritev',
156             'pwritev2',
157             'pwritev64',
158             'pwritev64v2',
159             'read',
160             'readahead',
161             'readv',
162             'recvfrom',
163             'sendfile',
164             'sendfile64',
165             'sendto',
166             'setsockopt',
167             'shutdown',
168             'signalfd4',
169             'splice',
170             'sync_file_range',
171             'syncfs',
172             'tee',
173             'utime',
174             'utimensat',
175             'vmsplice',
176             'write',
177             'writev']),
178     'bpf': set(['accept4',
179                 'acct',
180                 'bind',
181                 'bpf',
182                 'capget',
183                 'clone',
184                 'connect',
185                 'copy_file_range',
186                 'dup',
187                 'dup3',
188                 'epoll_create1',
189                 'epoll_ctl',
190                 'epoll_wait',
191                 'eventfd2',
```



```
192             'fallocate',
193             'fchdir',
194             'fchmod',
195             'fchown',
196             'fcntl',
197             'fcntl64',
198             'fdatasync',
199             'fgetxattr',
200             'flistxattr',
201             'flock',
202             'fork',
203             'fremovexattr',
204             'fsetxattr',
205             'fstatfs',
206             'fstatfs64',
207             'fsync',
208             'ftruncate',
209             'futimesat',
210             'get_robust_list',
211             'getdents',
212             'getdents64',
213             'getitimer',
214             'getpeername',
215             'getpgid',
216             'getppid',
217             'getpriority',
218             'getrusage',
219             'getsid',
220             'getsockname',
221             'getsockopt',
222             'inotify_add_watch',
223             'inotify_rm_watch',
224             'ioctl',
225             'ioperm',
226             'ioprio_get',
227             'ioprio_set',
228             'keyctl',
229             'kill',
230             'listen',
231             'llseek',
232             'lseek',
233             'memfd_create',
234             'migrate_pages',
235             'mmap_pgoff',
236             'move_pages',
237             'mq_getsetattr',
238             'mq_notify',
239             'mq_open',
240             'mq_timedreceive',
241             'mq_timedsend',
242             'msgrcv',
243             'old_readdir',
244             'open',
245             'open_by_handle_at',
246             'openat',
247             'perf_event_open',
248             'pipe2',
249             'prctl',
250             'pread64',
251             'preadv',
252             'preadv2',
253             'preadv64',
254             'preadv64v2',
255             'prlimit64',
256             'ptrace',
257             'pwrite64',
```

```
            'pwritev',
            'pwritev2',
            'pwritev64',
            'pwritev64v2',
            'read',
            'readahead',
            'readv',
            'recvfrom',
            'remap_file_pages',
            'rt_sigaction',
            'rt_sigprocmask',
            'rt_sigtimedwait',
            'sched_getaffinity',
            'sched_getattr',
            'sched_getparam',
            'sched_getscheduler',
            'sched_rr_get_interval',
            'sched_setaffinity',
            'sched_setattr',
            'sched_setparam',
            'sched_setscheduler',
            'semtimedop',
            'sendfile',
            'sendfile64',
            'sendto',
            'setitimer',
            'setns',
            'setpgid',
            'setpriority',
            'setsid',
            'setsockopt',
            'shmat',
            'shmctl',
            'shmdt',
            'shutdown',
            'sigaction',
            'sigaltstack',
            'signal',
            'signalfd4',
            'socket',
            'socketpair',
            'splice',
            'swapoff',
            'swapon',
            'sync_file_range',
            'syncfs',
            'tee',
            'umount',
            'uselib',
            'utime',
            'utimensat',
            'vfork',
            'vmsplice',
            'write',
            'writev']),
 'brk': set(['brk',
            'get_mempolicy',
            'getrusage',
            'io_cancel',
            'io_destroy',
            'io_getevents',
            'io_setup',
            'madvise',
            'mbind',
            'migrate_pages',
            'mincore',
```



```
            'mlockall',
            'modify_ldt',
            'move_pages',
            'mprotect',
            'mremap',
            'munlock',
            'munlockall',
            'pkey_mprotect',
            'prctl',
            'remap_file_pages',
            'shmdt',
            'swapoff']),
 'capset': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'clock_adjtime': set(['clock_adjtime',
                       'clock_getres',
                       'clock_gettime',
                       'clock_nanosleep',
                       'clock_settime',
                       'timer_create',
                       'timer_delete',
                       'timer_gettime',
```

```
                      'timer_settime']),
 'clock_nanosleep': set(['clock_adjtime',
                         'clock_getres',
                         'clock_gettime',
                         'clock_nanosleep',
                         'clock_settime',
                         'epoll_wait',
                         'faccessat',
                         'fchmod',
                         'fchmodat',
                         'fchown',
                         'fchownat',
                         'fstat',
                         'ftruncate',
                         'futex',
                         'futimesat',
                         'inotify_add_watch',
                         'io_getevents',
                         'ioctl',
                         'linkat',
                         'memfd_create',
                         'mq_getsetattr',
                         'mq_notify',
                         'mq_timedreceive',
                         'mq_timedsend',
                         'mq_unlink',
                         'nanosleep',
                         'newfstat',
                         'poll',
                         'ppoll',
                         'pselect6',
                         'readlinkat',
                         'recvmmsg',
                         'rt_sigtimedwait',
                         'sched_rr_get_interval',
                         'select',
                         'semtimedop',
                         'sendfile',
                         'sendfile64',
                         'settimeofday',
                         'stime',
                         'swapoff',
                         'swapon',
                         'timer_create',
                         'timer_delete',
                         'timer_gettime',
                         'timer_settime',
                         'timerfd_gettime',
                         'timerfd_settime',
                         'unlink',
                         'unlinkat',
                         'uselib',
                         'utime']),
 'clock_settime': set(['clock_adjtime',
                       'clock_getres',
                       'clock_gettime',
                       'clock_nanosleep',
                       'clock_settime',
                       'timer_create',
                       'timer_delete',
                       'timer_gettime',
                       'timer_settime']),
 'connect': set(['accept4',
                 'bind',
                 'bpf',
                 'connect',
```



```
                 'copy_file_range',
                 'epoll_ctl',
                 'epoll_wait',
                 'fallocate',
                 'fchdir',
                 'fchmod',
                 'fchown',
                 'fcntl',
                 'fcntl64',
                 'fdatasync',
                 'fgetxattr',
                 'flistxattr',
                 'flock',
                 'fremovexattr',
                 'fsetxattr',
                 'fstatfs',
                 'fstatfs64',
                 'fsync',
                 'ftruncate',
                 'futimesat',
                 'getdents',
                 'getdents64',
                 'getpeername',
                 'getsockname',
                 'getsockopt',
                 'inotify_add_watch',
                 'inotify_rm_watch',
                 'ioctl',
                 'listen',
                 'llseek',
                 'lseek',
                 'mq_getsetattr',
                 'mq_notify',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'old_readdir',
                 'perf_event_open',
                 'pread64',
                 'preadv',
                 'preadv2',
                 'preadv64',
                 'preadv64v2',
                 'pwrite64',
                 'pwritev',
                 'pwritev2',
                 'pwritev64',
                 'pwritev64v2',
                 'read',
                 'readahead',
                 'readv',
                 'recvfrom',
                 'sendfile',
                 'sendfile64',
                 'sendto',
                 'setsockopt',
                 'shutdown',
                 'signalfd4',
                 'splice',
                 'sync_file_range',
                 'syncfs',
                 'tee',
                 'utime',
                 'utimensat',
                 'vmsplice',
                 'write',
                 'writev']),
```

```
'copy_file_range': set(['accept4',
                        'bind',
                        'bpf',
                        'connect',
                        'copy_file_range',
                        'epoll_ctl',
                        'epoll_wait',
                        'fallocate',
                        'fchdir',
                        'fchmod',
                        'fchown',
                        'fcntl',
                        'fcntl64',
                        'fdatasync',
                        'fgetxattr',
                        'flistxattr',
                        'flock',
                        'fremovexattr',
                        'fsetxattr',
                        'fstatfs',
                        'fstatfs64',
                        'fsync',
                        'ftruncate',
                        'futimesat',
                        'getdents',
                        'getdents64',
                        'getpeername',
                        'getsockname',
                        'getsockopt',
                        'inotify_add_watch',
                        'inotify_rm_watch',
                        'ioctl',
                        'listen',
                        'llseek',
                        'lseek',
                        'mq_getsetattr',
                        'mq_notify',
                        'mq_timedreceive',
                        'mq_timedsend',
                        'old_readdir',
                        'perf_event_open',
                        'pread64',
                        'preadv',
                        'preadv2',
                        'preadv64',
                        'preadv64v2',
                        'pwrite64',
                        'pwritev',
                        'pwritev2',
                        'pwritev64',
                        'pwritev64v2',
                        'read',
                        'readahead',
                        'readv',
                        'recvfrom',
                        'sendfile',
                        'sendfile64',
                        'sendto',
                        'setsockopt',
                        'shutdown',
                        'signalfd4',
                        'splice',
                        'sync_file_range',
                        'syncfs',
                        'tee',
                        'utime',
```



```
                        'utimensat',
                        'vmsplice',
                        'write',
                        'writev']),
'delete_module': set(['delete_module', 'finit_module', 'init_module']),
'dmi_modalias_show': set(['dmi_modalias_show']),
'dup3': set(['dup2', 'dup3', 'select', 'unshare']),
'epoll_create1': set(['capget',
                      'capset',
                      'clone',
                      'epoll_create1',
                      'epoll_ctl',
                      'epoll_wait',
                      'faccessat',
                      'fork',
                      'get_robust_list',
                      'getgroups',
                      'getgroups16',
                      'getitimer',
                      'getpgid',
                      'getppid',
                      'getpriority',
                      'getresgid',
                      'getresgid16',
                      'getresuid',
                      'getresuid16',
                      'getrusage',
                      'getsid',
                      'ioprio_get',
                      'ioprio_set',
                      'keyctl',
                      'kill',
                      'migrate_pages',
                      'move_pages',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'msgrcv',
                      'perf_event_open',
                      'prctl',
                      'prlimit64',
                      'ptrace',
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setfsgid',
                      'setfsuid',
                      'setgid',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setregid',
                      'setresgid',
                      'setresuid',
                      'setreuid',
                      'setsid',
```

```
                    'setuid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'unshare',
                    'vfork']),
 'epoll_ctl': set(['accept4',
                   'acct',
                   'bind',
                   'bpf',
                   'brk',
                   'capget',
                   'clone',
                   'connect',
                   'copy_file_range',
                   'delete_module',
                   'dup',
                   'dup2',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'epoll_wait',
                   'eventfd2',
                   'exit_group',
                   'faccessat',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchmodat',
                   'fchown',
                   'fchownat',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'fgetxattr',
                   'finit_module',
                   'flistxattr',
                   'flock',
                   'fork',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'get_curr_temp',
                   'get_mempolicy',
                   'get_robust_list',
                   'get_trip_temp',
                   'getcwd',
                   'getdents',
                   'getdents64',
                   'getgroups',
                   'getgroups16',
                   'getitimer',
                   'getpeername',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'getsockname',
                   'getsockopt',
                   'init_module',
```



```
                   'inotify_add_watch',
                   'inotify_init1',
                   'inotify_rm_watch',
                   'io_cancel',
                   'io_destroy',
                   'io_getevents',
                   'io_setup',
                   'io_submit',
                   'ioctl',
                   'ioprio_get',
                   'ioprio_set',
                   'kexec_load',
                   'keyctl',
                   'kill',
                   'linkat',
                   'listen',
                   'llseek',
                   'lookup_dcookie',
                   'lseek',
                   'madvise',
                   'mbind',
                   'memfd_create',
                   'migrate_pages',
                   'mincore',
                   'mkdirat',
                   'mknodat',
                   'mlockall',
                   'mmap_pgoff',
                   'modify_ldt',
                   'move_pages',
                   'mprotect',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_open',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'mq_unlink',
                   'mremap',
                   'msgctl',
                   'msgrcv',
                   'msgsnd',
                   'munlock',
                   'munlockall',
                   'old_readdir',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'pivot_root',
                   'pkey_mprotect',
                   'prctl',
                   'pread64',
                   'preadv',
                   'preadv2',
                   'preadv64',
                   'preadv64v2',
                   'prlimit64',
                   'ptrace',
                   'pwrite64',
                   'pwritev',
                   'pwritev2',
                   'pwritev64',
                   'pwritev64v2',
                   'quotactl',
                   'read',
```

```
                    'readahead',
                    'readlinkat',
                    'readv',
                    'reboot',
                    'recvfrom',
                    'remap_file_pages',
                    'renameat2',
                    'request_key',
                    'rmdir',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'sched_yield',
                    'semctl',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'set_trip_temp',
                    'setgid',
                    'setgroups',
                    'setgroups16',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setregid',
                    'setresgid',
                    'setresuid',
                    'setreuid',
                    'setsid',
                    'setsockopt',
                    'setuid',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'symlinkat',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'timer_create',
                    'timer_delete',
                    'timer_getoverrun',
                    'timer_gettime',
                    'timer_settime',
                    'timerfd_create',
                    'timerfd_gettime',
```



```
                    'timerfd_settime',
                    'umount',
                    'unlink',
                    'unlinkat',
                    'unshare',
                    'uselib',
                    'ustat',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
                    'write',
                    'writev']),
 'epoll_wait': set(['accept4',
                    'acct',
                    'bind',
                    'bpf',
                    'capget',
                    'clone',
                    'connect',
                    'copy_file_range',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'epoll_wait',
                    'eventfd2',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fork',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'get_robust_list',
                    'getdents',
                    'getdents64',
                    'getitimer',
                    'getpeername',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'ioperm',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'listen',
```

```
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'migrate_pages',
                    'mmap_pgoff',
                    'move_pages',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'old_readdir',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'prctl',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'prlimit64',
                    'ptrace',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
```



```
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'umount',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
                    'write',
                    'writev']),
 'faccessat': set(['accept4',
                   'acct',
                   'capget',
                   'clone',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'flock',
                   'fork',
                   'get_robust_list',
                   'getcwd',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'lookup_dcookie',
                   'memfd_create',
                   'migrate_pages',
                   'mmap_pgoff',
                   'move_pages',
                   'mq_open',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'pivot_root',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'quotactl',
                   'remap_file_pages',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
```

```
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setresuid',
                    'setreuid',
                    'setsid',
                    'setuid',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'umount',
                    'unshare',
                    'uselib',
                    'vfork']),
 'fallocate': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
```



```
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'fchdir': set(['accept4',
                'acct',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getcwd',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
```

```
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lookup_dcookie',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'quotactl',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'unshare',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchmod': set(['accept4',
                'acct',
                'bind',
```



```
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getcwd',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lookup_dcookie',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'quotactl',
                'read',
                'readahead',
```

```
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'unshare',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchmodat': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'getcwd',
                  'lookup_dcookie',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pivot_root',
                  'quotactl',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'unshare',
                  'uselib']),
 'fchown': set(['accept4',
                'acct',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
```



```
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getcwd',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lookup_dcookie',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'quotactl',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
```

```
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'unshare',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchownat': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'getcwd',
                  'lookup_dcookie',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pivot_root',
                  'quotactl',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'unshare',
                  'uselib']),
 'fcntl': set(['accept4',
               'acct',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'epoll_wait',
```



```
               'eventfd2',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'memfd_create',
               'mmap_pgoff',
               'mq_getsetattr',
               'mq_notify',
               'mq_open',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'open',
               'open_by_handle_at',
               'openat',
               'perf_event_open',
               'pipe2',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'remap_file_pages',
               'sendfile',
               'sendfile64',
               'sendto',
               'setns',
               'setsockopt',
               'shmat',
               'shmctl',
               'shmdt',
               'shutdown',
```

```
1578                 'signalfd4',
1579                 'socket',
1580                 'socketpair',
1581                 'splice',
1582                 'swapoff',
1583                 'swapon',
1584                 'sync_file_range',
1585                 'syncfs',
1586                 'tee',
1587                 'uselib',
1588                 'utime',
1589                 'utimensat',
1590                 'vmsplice',
1591                 'write',
1592                 'writev']),
1593  'fcntl64': set(['accept4',
1594                  'acct',
1595                  'bind',
1596                  'bpf',
1597                  'connect',
1598                  'copy_file_range',
1599                  'dup',
1600                  'dup3',
1601                  'epoll_create1',
1602                  'epoll_ctl',
1603                  'epoll_wait',
1604                  'eventfd2',
1605                  'fallocate',
1606                  'fchdir',
1607                  'fchmod',
1608                  'fchown',
1609                  'fcntl',
1610                  'fcntl64',
1611                  'fdatasync',
1612                  'fgetxattr',
1613                  'flistxattr',
1614                  'flock',
1615                  'fremovexattr',
1616                  'fsetxattr',
1617                  'fstatfs',
1618                  'fstatfs64',
1619                  'fsync',
1620                  'ftruncate',
1621                  'futimesat',
1622                  'getdents',
1623                  'getdents64',
1624                  'getpeername',
1625                  'getsockname',
1626                  'getsockopt',
1627                  'inotify_add_watch',
1628                  'inotify_rm_watch',
1629                  'ioctl',
1630                  'listen',
1631                  'llseek',
1632                  'lseek',
1633                  'memfd_create',
1634                  'mmap_pgoff',
1635                  'mq_getsetattr',
1636                  'mq_notify',
1637                  'mq_open',
1638                  'mq_timedreceive',
1639                  'mq_timedsend',
1640                  'old_readdir',
1641                  'open',
1642                  'open_by_handle_at',
1643                  'openat',
```



```
1644                  'perf_event_open',
1645                  'pipe2',
1646                  'pread64',
1647                  'preadv',
1648                  'preadv2',
1649                  'preadv64',
1650                  'preadv64v2',
1651                  'pwrite64',
1652                  'pwritev',
1653                  'pwritev2',
1654                  'pwritev64',
1655                  'pwritev64v2',
1656                  'read',
1657                  'readahead',
1658                  'readv',
1659                  'recvfrom',
1660                  'remap_file_pages',
1661                  'sendfile',
1662                  'sendfile64',
1663                  'sendto',
1664                  'setns',
1665                  'setsockopt',
1666                  'shmat',
1667                  'shmctl',
1668                  'shmdt',
1669                  'shutdown',
1670                  'signalfd4',
1671                  'socket',
1672                  'socketpair',
1673                  'splice',
1674                  'swapoff',
1675                  'swapon',
1676                  'sync_file_range',
1677                  'syncfs',
1678                  'tee',
1679                  'uselib',
1680                  'utime',
1681                  'utimensat',
1682                  'vmsplice',
1683                  'write',
1684                  'writev']),
1685  'fdatasync': set(['accept4',
1686                    'bind',
1687                    'bpf',
1688                    'connect',
1689                    'copy_file_range',
1690                    'epoll_ctl',
1691                    'epoll_wait',
1692                    'fallocate',
1693                    'fchdir',
1694                    'fchmod',
1695                    'fchown',
1696                    'fcntl',
1697                    'fcntl64',
1698                    'fdatasync',
1699                    'fgetxattr',
1700                    'flistxattr',
1701                    'flock',
1702                    'fremovexattr',
1703                    'fsetxattr',
1704                    'fstatfs',
1705                    'fstatfs64',
1706                    'fsync',
1707                    'ftruncate',
1708                    'futimesat',
1709                    'getdents',
```

```
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'fgetxattr': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
```



```
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'finit_module': set(['delete_module', 'finit_module', 'init_module']),
 'flistxattr': set(['accept4',
                     'bind',
                     'bpf',
                     'connect',
                     'copy_file_range',
                     'epoll_ctl',
                     'epoll_wait',
                     'fallocate',
                     'fchdir',
                     'fchmod',
                     'fchown',
                     'fcntl',
                     'fcntl64',
                     'fdatasync',
                     'fgetxattr',
                     'flistxattr',
```

```
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'flock': set(['accept4',
               'acct',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'epoll_wait',
               'eventfd2',
```



```
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'memfd_create',
               'mmap_pgoff',
               'mq_getsetattr',
               'mq_notify',
               'mq_open',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'open',
               'open_by_handle_at',
               'openat',
               'perf_event_open',
               'pipe2',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'quotactl',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'remap_file_pages',
               'sendfile',
               'sendfile64',
               'sendto',
               'setns',
               'setsockopt',
               'shmat',
               'shmctl',
               'shmdt',
               'shutdown',
```

```
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'umount',
                'uselib',
                'ustat',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fremovexattr': set(['accept4',
                      'bind',
                      'bpf',
                      'connect',
                      'copy_file_range',
                      'epoll_ctl',
                      'epoll_wait',
                      'fallocate',
                      'fchdir',
                      'fchmod',
                      'fchown',
                      'fcntl',
                      'fcntl64',
                      'fdatasync',
                      'fgetxattr',
                      'flistxattr',
                      'flock',
                      'fremovexattr',
                      'fsetxattr',
                      'fstatfs',
                      'fstatfs64',
                      'fsync',
                      'ftruncate',
                      'futimesat',
                      'getdents',
                      'getdents64',
                      'getpeername',
                      'getsockname',
                      'getsockopt',
                      'inotify_add_watch',
                      'inotify_rm_watch',
                      'ioctl',
                      'listen',
                      'llseek',
                      'lseek',
                      'mq_getsetattr',
                      'mq_notify',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'old_readdir',
                      'perf_event_open',
                      'pread64',
                      'preadv',
                      'preadv2',
                      'preadv64',
                      'preadv64v2',
                      'pwrite64',
                      'pwritev',
                      'pwritev2',
```



```
                      'pwritev64',
                      'pwritev64v2',
                      'read',
                      'readahead',
                      'readv',
                      'recvfrom',
                      'sendfile',
                      'sendfile64',
                      'sendto',
                      'setsockopt',
                      'shutdown',
                      'signalfd4',
                      'splice',
                      'sync_file_range',
                      'syncfs',
                      'tee',
                      'utime',
                      'utimensat',
                      'vmsplice',
                      'write',
                      'writev']),
 'fsetxattr': set(['accept4',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'epoll_ctl',
                   'epoll_wait',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'fgetxattr',
                   'flistxattr',
                   'flock',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'getdents',
                   'getdents64',
                   'getpeername',
                   'getsockname',
                   'getsockopt',
                   'inotify_add_watch',
                   'inotify_rm_watch',
                   'ioctl',
                   'listen',
                   'llseek',
                   'lseek',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'old_readdir',
                   'perf_event_open',
                   'pread64',
                   'preadv',
                   'preadv2',
                   'preadv64',
```

```
                  'preadv64v2',
                  'pwrite64',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setsockopt',
                  'shutdown',
                  'signalfd4',
                  'splice',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'utime',
                  'utimensat',
                  'vmsplice',
                  'write',
                  'writev']),
 'fstat': set(['fstat', 'lstat', 'newfstat', 'stat']),
 'fstatfs': set(['accept4',
                 'bind',
                 'bpf',
                 'connect',
                 'copy_file_range',
                 'epoll_ctl',
                 'epoll_wait',
                 'fallocate',
                 'fchdir',
                 'fchmod',
                 'fchown',
                 'fcntl',
                 'fcntl64',
                 'fdatasync',
                 'fgetxattr',
                 'flistxattr',
                 'flock',
                 'fremovexattr',
                 'fsetxattr',
                 'fstatfs',
                 'fstatfs64',
                 'fsync',
                 'ftruncate',
                 'futimesat',
                 'getdents',
                 'getdents64',
                 'getpeername',
                 'getsockname',
                 'getsockopt',
                 'inotify_add_watch',
                 'inotify_rm_watch',
                 'ioctl',
                 'listen',
                 'llseek',
                 'lseek',
                 'mq_getsetattr',
                 'mq_notify',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'old_readdir',
```



```
                 'perf_event_open',
                 'pread64',
                 'preadv',
                 'preadv2',
                 'preadv64',
                 'preadv64v2',
                 'pwrite64',
                 'pwritev',
                 'pwritev2',
                 'pwritev64',
                 'pwritev64v2',
                 'read',
                 'readahead',
                 'readv',
                 'recvfrom',
                 'sendfile',
                 'sendfile64',
                 'sendto',
                 'setsockopt',
                 'shutdown',
                 'signalfd4',
                 'splice',
                 'statfs',
                 'statfs64',
                 'sync_file_range',
                 'syncfs',
                 'tee',
                 'ustat',
                 'utime',
                 'utimensat',
                 'vmsplice',
                 'write',
                 'writev']),
 'fstatfs64': set(['accept4',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'epoll_ctl',
                   'epoll_wait',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'fgetxattr',
                   'flistxattr',
                   'flock',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'getdents',
                   'getdents64',
                   'getpeername',
                   'getsockname',
                   'getsockopt',
                   'inotify_add_watch',
                   'inotify_rm_watch',
                   'ioctl',
                   'listen',
```

```
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'statfs',
                    'statfs64',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'ustat',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'fsync': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
```



```
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'sendfile',
               'sendfile64',
               'sendto',
               'setsockopt',
               'shutdown',
               'signalfd4',
               'splice',
               'sync_file_range',
               'syncfs',
               'tee',
               'utime',
               'utimensat',
               'vmsplice',
               'write',
               'writev']),
 'ftruncate': set(['accept4',
                   'acct',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'epoll_wait',
                   'eventfd2',
                   'faccessat',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchmodat',
                   'fchown',
                   'fchownat',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
```

```
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'linkat',
                    'listen',
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'mq_unlink',
                    'old_readdir',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readlinkat',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setns',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
```



```
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'unlink',
                    'unlinkat',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'futex': set(['clock_nanosleep',
               'epoll_wait',
               'faccessat',
               'fchmod',
               'fchmodat',
               'fchown',
               'fchownat',
               'fstat',
               'ftruncate',
               'futex',
               'futimesat',
               'inotify_add_watch',
               'io_getevents',
               'ioctl',
               'linkat',
               'memfd_create',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'mq_unlink',
               'nanosleep',
               'newfstat',
               'poll',
               'ppoll',
               'pselect6',
               'readlinkat',
               'recvmmsg',
               'rt_sigtimedwait',
               'sched_rr_get_interval',
               'select',
               'semtimedop',
               'sendfile',
               'sendfile64',
               'settimeofday',
               'stime',
               'swapoff',
               'swapon',
               'timer_gettime',
               'timer_settime',
               'timerfd_gettime',
               'timerfd_settime',
               'unlink',
               'unlinkat',
               'uselib',
               'utime']),
 'futimesat': set(['accept4',
                   'adjtimex',
                   'alarm',
                   'bind',
                   'bpf',
                   'clock_adjtime',
                   'clock_nanosleep',
                   'connect',
```

```
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'faccessat',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchmodat',
                    'fchown',
                    'fchownat',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstat',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futex',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getitimer',
                    'getpeername',
                    'getrusage',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'io_getevents',
                    'ioctl',
                    'linkat',
                    'listen',
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'mq_unlink',
                    'nanosleep',
                    'newfstat',
                    'old_readdir',
                    'perf_event_open',
                    'poll',
                    'ppoll',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pselect6',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readlinkat',
```



```
                    'readv',
                    'recvfrom',
                    'recvmmsg',
                    'rt_sigtimedwait',
                    'sched_rr_get_interval',
                    'select',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setitimer',
                    'setsockopt',
                    'settimeofday',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'stime',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'timer_gettime',
                    'timer_settime',
                    'timerfd_gettime',
                    'timerfd_settime',
                    'unlink',
                    'unlinkat',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'wait4',
                    'waitid',
                    'write',
                    'writev']),
    'get_mempolicy': set(['brk',
                          'capget',
                          'clone',
                          'fork',
                          'get_mempolicy',
                          'get_robust_list',
                          'getitimer',
                          'getpgid',
                          'getppid',
                          'getpriority',
                          'getrusage',
                          'getsid',
                          'ioprio_get',
                          'ioprio_set',
                          'keyctl',
                          'kill',
                          'madvise',
                          'mbind',
                          'migrate_pages',
                          'mincore',
                          'mlockall',
                          'move_pages',
                          'mprotect',
                          'mq_timedreceive',
                          'mq_timedsend',
                          'mremap',
                          'msgrcv',
                          'munlock',
                          'munlockall',
                          'perf_event_open',
```

```
                      'pkey_mprotect',
                      'prctl',
                      'prlimit64',
                      'ptrace',
                      'remap_file_pages',
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'set_mempolicy',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setsid',
                      'shmdt',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'umount',
                      'vfork']),
 'getcwd': set(['accept4',
                'acct',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'eventfd2',
                'flock',
                'ftruncate',
                'getcwd',
                'linkat',
                'lookup_dcookie',
                'memfd_create',
                'mkdirat',
                'mknodat',
                'mmap_pgoff',
                'mq_open',
                'mq_unlink',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'quotactl',
                'remap_file_pages',
                'renameat2',
                'rmdir',
                'setns',
                'shmat',
                'shmctl',
                'shmdt',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
```



```
                'symlinkat',
                'umount',
                'unlink',
                'unlinkat',
                'unshare',
                'uselib']),
 'getdents': set(['accept4',
                  'bind',
                  'bpf',
                  'capget',
                  'clone',
                  'connect',
                  'copy_file_range',
                  'epoll_ctl',
                  'epoll_wait',
                  'fallocate',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
                  'fork',
                  'fremovexattr',
                  'fsetxattr',
                  'fstatfs',
                  'fstatfs64',
                  'fsync',
                  'ftruncate',
                  'futimesat',
                  'get_robust_list',
                  'getdents',
                  'getdents64',
                  'getitimer',
                  'getpeername',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'inotify_rm_watch',
                  'ioctl',
                  'ioperm',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'listen',
                  'llseek',
                  'lseek',
                  'migrate_pages',
                  'move_pages',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'old_readdir',
                  'perf_event_open',
                  'prctl',
```

```
                  'pread64',
                  'preadv',
                  'preadv2',
                  'preadv64',
                  'preadv64v2',
                  'prlimit64',
                  'ptrace',
                  'pwrite64',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'setsockopt',
                  'shutdown',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'signalfd4',
                  'splice',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'umount',
                  'utime',
                  'utimensat',
                  'vfork',
                  'vmsplice',
                  'write',
                  'writev']),
  'getdents64': set(['accept4',
                     'bind',
                     'bpf',
                     'capget',
                     'clone',
                     'connect',
                     'copy_file_range',
                     'epoll_ctl',
                     'epoll_wait',
                     'fallocate',
                     'fchdir',
                     'fchmod',
```



```
                     'fchown',
                     'fcntl',
                     'fcntl64',
                     'fdatasync',
                     'fgetxattr',
                     'flistxattr',
                     'flock',
                     'fork',
                     'fremovexattr',
                     'fsetxattr',
                     'fstatfs',
                     'fstatfs64',
                     'fsync',
                     'ftruncate',
                     'futimesat',
                     'get_robust_list',
                     'getdents',
                     'getdents64',
                     'getitimer',
                     'getpeername',
                     'getpgid',
                     'getppid',
                     'getpriority',
                     'getrusage',
                     'getsid',
                     'getsockname',
                     'getsockopt',
                     'inotify_add_watch',
                     'inotify_rm_watch',
                     'ioctl',
                     'ioperm',
                     'ioprio_get',
                     'ioprio_set',
                     'keyctl',
                     'kill',
                     'listen',
                     'llseek',
                     'lseek',
                     'migrate_pages',
                     'move_pages',
                     'mq_getsetattr',
                     'mq_notify',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'msgrcv',
                     'old_readdir',
                     'perf_event_open',
                     'prctl',
                     'pread64',
                     'preadv',
                     'preadv2',
                     'preadv64',
                     'preadv64v2',
                     'prlimit64',
                     'ptrace',
                     'pwrite64',
                     'pwritev',
                     'pwritev2',
                     'pwritev64',
                     'pwritev64v2',
                     'read',
                     'readahead',
                     'readv',
                     'recvfrom',
                     'rt_sigaction',
                     'rt_sigprocmask',
```

```
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'setsockopt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'umount',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
                    'write',
                    'writev']),
 'getegid': set(['capget',
                 'clone',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'migrate_pages',
                 'move_pages',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'msgrcv',
                 'perf_event_open',
                 'prctl',
                 'prlimit64',
                 'ptrace',
                 'rt_sigaction',
                 'rt_sigprocmask',
                 'rt_sigtimedwait',
                 'sched_getaffinity',
                 'sched_getattr',
                 'sched_getparam',
                 'sched_getscheduler',
```



```
                 'sched_rr_get_interval',
                 'sched_setaffinity',
                 'sched_setattr',
                 'sched_setparam',
                 'sched_setscheduler',
                 'semtimedop',
                 'setitimer',
                 'setns',
                 'setpgid',
                 'setpriority',
                 'setsid',
                 'sigaction',
                 'sigaltstack',
                 'signal',
                 'umount',
                 'vfork']),
 'getegid16': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'geteuid': set(['capget',
                 'clone',
                 'fork',
                 'get_robust_list',
```

```
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'geteuid16': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
```



```
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'getgid': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
```

```
3162                 'umount',
3163                 'vfork']),
3164  'getgid16': set(['capget',
3165                   'clone',
3166                   'fork',
3167                   'get_robust_list',
3168                   'getitimer',
3169                   'getpgid',
3170                   'getppid',
3171                   'getpriority',
3172                   'getrusage',
3173                   'getsid',
3174                   'ioprio_get',
3175                   'ioprio_set',
3176                   'keyctl',
3177                   'kill',
3178                   'migrate_pages',
3179                   'move_pages',
3180                   'mq_timedreceive',
3181                   'mq_timedsend',
3182                   'msgrcv',
3183                   'perf_event_open',
3184                   'prctl',
3185                   'prlimit64',
3186                   'ptrace',
3187                   'rt_sigaction',
3188                   'rt_sigprocmask',
3189                   'rt_sigtimedwait',
3190                   'sched_getaffinity',
3191                   'sched_getattr',
3192                   'sched_getparam',
3193                   'sched_getscheduler',
3194                   'sched_rr_get_interval',
3195                   'sched_setaffinity',
3196                   'sched_setattr',
3197                   'sched_setparam',
3198                   'sched_setscheduler',
3199                   'semtimedop',
3200                   'setitimer',
3201                   'setns',
3202                   'setpgid',
3203                   'setpriority',
3204                   'setsid',
3205                   'sigaction',
3206                   'sigaltstack',
3207                   'signal',
3208                   'umount',
3209                   'vfork']),
3210  'getgroups': set(['capget',
3211                    'capset',
3212                    'clone',
3213                    'epoll_create1',
3214                    'faccessat',
3215                    'fork',
3216                    'get_robust_list',
3217                    'getgroups',
3218                    'getgroups16',
3219                    'getitimer',
3220                    'getpgid',
3221                    'getppid',
3222                    'getpriority',
3223                    'getresgid',
3224                    'getresgid16',
3225                    'getresuid',
3226                    'getresuid16',
3227                    'getrusage',
```



```
3228                    'getsid',
3229                    'ioprio_get',
3230                    'ioprio_set',
3231                    'keyctl',
3232                    'kill',
3233                    'migrate_pages',
3234                    'move_pages',
3235                    'mq_timedreceive',
3236                    'mq_timedsend',
3237                    'msgrcv',
3238                    'perf_event_open',
3239                    'prctl',
3240                    'prlimit64',
3241                    'ptrace',
3242                    'rt_sigaction',
3243                    'rt_sigprocmask',
3244                    'rt_sigtimedwait',
3245                    'sched_getaffinity',
3246                    'sched_getattr',
3247                    'sched_getparam',
3248                    'sched_getscheduler',
3249                    'sched_rr_get_interval',
3250                    'sched_setaffinity',
3251                    'sched_setattr',
3252                    'sched_setparam',
3253                    'sched_setscheduler',
3254                    'semtimedop',
3255                    'setfsgid',
3256                    'setfsuid',
3257                    'setgid',
3258                    'setitimer',
3259                    'setns',
3260                    'setpgid',
3261                    'setpriority',
3262                    'setregid',
3263                    'setresgid',
3264                    'setresuid',
3265                    'setreuid',
3266                    'setsid',
3267                    'setuid',
3268                    'sigaction',
3269                    'sigaltstack',
3270                    'signal',
3271                    'umount',
3272                    'unshare',
3273                    'vfork']),
3274  'getgroups16': set(['capget',
3275                      'capset',
3276                      'clone',
3277                      'epoll_create1',
3278                      'faccessat',
3279                      'fork',
3280                      'get_robust_list',
3281                      'getgroups',
3282                      'getgroups16',
3283                      'getitimer',
3284                      'getpgid',
3285                      'getppid',
3286                      'getpriority',
3287                      'getresgid',
3288                      'getresgid16',
3289                      'getresuid',
3290                      'getresuid16',
3291                      'getrusage',
3292                      'getsid',
3293                      'ioprio_get',
```

```
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setfsgid',
                    'setfsuid',
                    'setgid',
                    'setgroups',
                    'setgroups16',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setregid',
                    'setresgid',
                    'setresuid',
                    'setreuid',
                    'setsid',
                    'setuid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'unshare',
                    'vfork']),
 'getitimer': set(['exit_group', 'setitimer', 'timer_create']),
 'getpeername': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
```



```
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'getppid': set(['capget',
                 'clone',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'migrate_pages',
```

```
3426                 'move_pages',
3427                 'mq_timedreceive',
3428                 'mq_timedsend',
3429                 'msgrcv',
3430                 'perf_event_open',
3431                 'prctl',
3432                 'prlimit64',
3433                 'ptrace',
3434                 'rt_sigaction',
3435                 'rt_sigprocmask',
3436                 'rt_sigtimedwait',
3437                 'sched_getaffinity',
3438                 'sched_getattr',
3439                 'sched_getparam',
3440                 'sched_getscheduler',
3441                 'sched_rr_get_interval',
3442                 'sched_setaffinity',
3443                 'sched_setattr',
3444                 'sched_setparam',
3445                 'sched_setscheduler',
3446                 'semtimedop',
3447                 'setitimer',
3448                 'setns',
3449                 'setpgid',
3450                 'setpriority',
3451                 'setsid',
3452                 'sigaction',
3453                 'sigaltstack',
3454                 'signal',
3455                 'umount',
3456                 'vfork']),
3457  'getpriority': set(['capget',
3458                      'clone',
3459                      'fork',
3460                      'get_robust_list',
3461                      'getitimer',
3462                      'getpgid',
3463                      'getppid',
3464                      'getpriority',
3465                      'getrusage',
3466                      'getsid',
3467                      'ioprio_get',
3468                      'ioprio_set',
3469                      'keyctl',
3470                      'kill',
3471                      'migrate_pages',
3472                      'move_pages',
3473                      'mq_timedreceive',
3474                      'mq_timedsend',
3475                      'msgrcv',
3476                      'perf_event_open',
3477                      'prctl',
3478                      'prlimit64',
3479                      'ptrace',
3480                      'rt_sigaction',
3481                      'rt_sigprocmask',
3482                      'rt_sigtimedwait',
3483                      'sched_getaffinity',
3484                      'sched_getattr',
3485                      'sched_getparam',
3486                      'sched_getscheduler',
3487                      'sched_rr_get_interval',
3488                      'sched_setaffinity',
3489                      'sched_setattr',
3490                      'sched_setparam',
3491                      'sched_setscheduler',
```



```
3492                      'semtimedop',
3493                      'setitimer',
3494                      'setns',
3495                      'setpgid',
3496                      'setpriority',
3497                      'setresuid',
3498                      'setreuid',
3499                      'setsid',
3500                      'setuid',
3501                      'sigaction',
3502                      'sigaltstack',
3503                      'signal',
3504                      'umount',
3505                      'vfork']),
3506  'getresgid': set(['capget',
3507                    'clone',
3508                    'fork',
3509                    'get_robust_list',
3510                    'getitimer',
3511                    'getpgid',
3512                    'getppid',
3513                    'getpriority',
3514                    'getrusage',
3515                    'getsid',
3516                    'ioprio_get',
3517                    'ioprio_set',
3518                    'keyctl',
3519                    'kill',
3520                    'migrate_pages',
3521                    'move_pages',
3522                    'mq_timedreceive',
3523                    'mq_timedsend',
3524                    'msgrcv',
3525                    'perf_event_open',
3526                    'prctl',
3527                    'prlimit64',
3528                    'ptrace',
3529                    'rt_sigaction',
3530                    'rt_sigprocmask',
3531                    'rt_sigtimedwait',
3532                    'sched_getaffinity',
3533                    'sched_getattr',
3534                    'sched_getparam',
3535                    'sched_getscheduler',
3536                    'sched_rr_get_interval',
3537                    'sched_setaffinity',
3538                    'sched_setattr',
3539                    'sched_setparam',
3540                    'sched_setscheduler',
3541                    'semtimedop',
3542                    'setitimer',
3543                    'setns',
3544                    'setpgid',
3545                    'setpriority',
3546                    'setsid',
3547                    'sigaction',
3548                    'sigaltstack',
3549                    'signal',
3550                    'umount',
3551                    'vfork']),
3552  'getresgid16': set(['capget',
3553                      'clone',
3554                      'fork',
3555                      'get_robust_list',
3556                      'getitimer',
3557                      'getpgid',
```

```
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'getresuid': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
```



```
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'getresuid16': set(['capget',
                     'clone',
                     'fork',
                     'get_robust_list',
                     'getitimer',
                     'getpgid',
                     'getppid',
                     'getpriority',
                     'getrusage',
                     'getsid',
                     'ioprio_get',
                     'ioprio_set',
                     'keyctl',
                     'kill',
                     'migrate_pages',
                     'move_pages',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'msgrcv',
                     'perf_event_open',
                     'prctl',
                     'prlimit64',
                     'ptrace',
                     'rt_sigaction',
                     'rt_sigprocmask',
                     'rt_sigtimedwait',
                     'sched_getaffinity',
                     'sched_getattr',
                     'sched_getparam',
                     'sched_getscheduler',
                     'sched_rr_get_interval',
                     'sched_setaffinity',
                     'sched_setattr',
                     'sched_setparam',
                     'sched_setscheduler',
                     'semtimedop',
                     'setitimer',
                     'setns',
                     'setpgid',
                     'setpriority',
                     'setsid',
                     'sigaction',
                     'sigaltstack',
                     'signal',
                     'umount',
                     'vfork']),
```

```
3690  'getrlimit': set(['old_getrlimit', 'prlimit64', 'setrlimit']),
3691  'getrusage': set(['exit_group', 'timer_create']),
3692  'getsockname': set(['accept4',
3693                      'bind',
3694                      'bpf',
3695                      'connect',
3696                      'copy_file_range',
3697                      'epoll_ctl',
3698                      'epoll_wait',
3699                      'fallocate',
3700                      'fchdir',
3701                      'fchmod',
3702                      'fchown',
3703                      'fcntl',
3704                      'fcntl64',
3705                      'fdatasync',
3706                      'fgetxattr',
3707                      'flistxattr',
3708                      'flock',
3709                      'fremovexattr',
3710                      'fsetxattr',
3711                      'fstatfs',
3712                      'fstatfs64',
3713                      'fsync',
3714                      'ftruncate',
3715                      'futimesat',
3716                      'getdents',
3717                      'getdents64',
3718                      'getpeername',
3719                      'getsockname',
3720                      'getsockopt',
3721                      'inotify_add_watch',
3722                      'inotify_rm_watch',
3723                      'ioctl',
3724                      'listen',
3725                      'llseek',
3726                      'lseek',
3727                      'mq_getsetattr',
3728                      'mq_notify',
3729                      'mq_timedreceive',
3730                      'mq_timedsend',
3731                      'old_readdir',
3732                      'perf_event_open',
3733                      'pread64',
3734                      'preadv',
3735                      'preadv2',
3736                      'preadv64',
3737                      'preadv64v2',
3738                      'pwrite64',
3739                      'pwritev',
3740                      'pwritev2',
3741                      'pwritev64',
3742                      'pwritev64v2',
3743                      'read',
3744                      'readahead',
3745                      'readv',
3746                      'recvfrom',
3747                      'sendfile',
3748                      'sendfile64',
3749                      'sendto',
3750                      'setsockopt',
3751                      'shutdown',
3752                      'signalfd4',
3753                      'splice',
3754                      'sync_file_range',
3755                      'syncfs',
```



```
3756                      'tee',
3757                      'utime',
3758                      'utimensat',
3759                      'vmsplice',
3760                      'write',
3761                      'writev']),
3762  'getsockopt': set(['accept4',
3763                     'bind',
3764                     'bpf',
3765                     'connect',
3766                     'copy_file_range',
3767                     'epoll_ctl',
3768                     'epoll_wait',
3769                     'fallocate',
3770                     'fchdir',
3771                     'fchmod',
3772                     'fchown',
3773                     'fcntl',
3774                     'fcntl64',
3775                     'fdatasync',
3776                     'fgetxattr',
3777                     'flistxattr',
3778                     'flock',
3779                     'fremovexattr',
3780                     'fsetxattr',
3781                     'fstatfs',
3782                     'fstatfs64',
3783                     'fsync',
3784                     'ftruncate',
3785                     'futimesat',
3786                     'getdents',
3787                     'getdents64',
3788                     'getpeername',
3789                     'getsockname',
3790                     'getsockopt',
3791                     'inotify_add_watch',
3792                     'inotify_rm_watch',
3793                     'ioctl',
3794                     'listen',
3795                     'llseek',
3796                     'lseek',
3797                     'mq_getsetattr',
3798                     'mq_notify',
3799                     'mq_timedreceive',
3800                     'mq_timedsend',
3801                     'old_readdir',
3802                     'perf_event_open',
3803                     'pread64',
3804                     'preadv',
3805                     'preadv2',
3806                     'preadv64',
3807                     'preadv64v2',
3808                     'pwrite64',
3809                     'pwritev',
3810                     'pwritev2',
3811                     'pwritev64',
3812                     'pwritev64v2',
3813                     'read',
3814                     'readahead',
3815                     'readv',
3816                     'recvfrom',
3817                     'sendfile',
3818                     'sendfile64',
3819                     'sendto',
3820                     'setsockopt',
3821                     'shutdown',
```

```
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'getuid': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'getuid16': set(['capget',
                  'clone',
                  'fork',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
```



```
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'getxattr': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'getcwd',
                  'lookup_dcookie',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pivot_root',
                  'quotactl',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'unshare',
```

```
                    'uselib']),
 'init_module': set(['delete_module', 'finit_module', 'init_module']),
 'inotify_add_watch': set(['accept4',
                           'acct',
                           'bind',
                           'bpf',
                           'connect',
                           'copy_file_range',
                           'dup',
                           'dup3',
                           'epoll_create1',
                           'epoll_ctl',
                           'epoll_wait',
                           'eventfd2',
                           'fallocate',
                           'fchdir',
                           'fchmod',
                           'fchown',
                           'fcntl',
                           'fcntl64',
                           'fdatasync',
                           'fgetxattr',
                           'flistxattr',
                           'flock',
                           'fremovexattr',
                           'fsetxattr',
                           'fstatfs',
                           'fstatfs64',
                           'fsync',
                           'ftruncate',
                           'futimesat',
                           'getcwd',
                           'getdents',
                           'getdents64',
                           'getpeername',
                           'getsockname',
                           'getsockopt',
                           'inotify_add_watch',
                           'inotify_rm_watch',
                           'ioctl',
                           'linkat',
                           'listen',
                           'llseek',
                           'lseek',
                           'memfd_create',
                           'mkdirat',
                           'mknodat',
                           'mmap_pgoff',
                           'mq_getsetattr',
                           'mq_notify',
                           'mq_open',
                           'mq_timedreceive',
                           'mq_timedsend',
                           'mq_unlink',
                           'old_readdir',
                           'open',
                           'open_by_handle_at',
                           'openat',
                           'perf_event_open',
                           'pipe2',
                           'pivot_root',
                           'pread64',
                           'preadv',
                           'preadv2',
                           'preadv64',
                           'preadv64v2',
```



```
                           'pwrite64',
                           'pwritev',
                           'pwritev2',
                           'pwritev64',
                           'pwritev64v2',
                           'read',
                           'readahead',
                           'readv',
                           'recvfrom',
                           'remap_file_pages',
                           'renameat2',
                           'rmdir',
                           'sendfile',
                           'sendfile64',
                           'sendto',
                           'setns',
                           'setsockopt',
                           'shmat',
                           'shmctl',
                           'shmdt',
                           'shutdown',
                           'signalfd4',
                           'socket',
                           'socketpair',
                           'splice',
                           'swapoff',
                           'swapon',
                           'symlinkat',
                           'sync_file_range',
                           'syncfs',
                           'tee',
                           'unlink',
                           'unlinkat',
                           'uselib',
                           'utime',
                           'utimensat',
                           'vmsplice',
                           'write',
                           'writev']),
 'inotify_init1': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'inotify_add_watch',
                       'inotify_init1',
                       'inotify_rm_watch',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'migrate_pages',
                       'move_pages',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'msgrcv',
                       'perf_event_open',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'rt_sigaction',
```

```
4086                        'rt_sigprocmask',
4087                        'rt_sigtimedwait',
4088                        'sched_getaffinity',
4089                        'sched_getattr',
4090                        'sched_getparam',
4091                        'sched_getscheduler',
4092                        'sched_rr_get_interval',
4093                        'sched_setaffinity',
4094                        'sched_setattr',
4095                        'sched_setparam',
4096                        'sched_setscheduler',
4097                        'semtimedop',
4098                        'setitimer',
4099                        'setns',
4100                        'setpgid',
4101                        'setpriority',
4102                        'setsid',
4103                        'sigaction',
4104                        'sigaltstack',
4105                        'signal',
4106                        'umount',
4107                        'vfork']),
4108  'inotify_rm_watch': set(['accept4',
4109                           'acct',
4110                           'bind',
4111                           'bpf',
4112                           'connect',
4113                           'copy_file_range',
4114                           'dup',
4115                           'dup3',
4116                           'epoll_create1',
4117                           'epoll_ctl',
4118                           'epoll_wait',
4119                           'eventfd2',
4120                           'fallocate',
4121                           'fchdir',
4122                           'fchmod',
4123                           'fchown',
4124                           'fcntl',
4125                           'fcntl64',
4126                           'fdatasync',
4127                           'fgetxattr',
4128                           'flistxattr',
4129                           'flock',
4130                           'fremovexattr',
4131                           'fsetxattr',
4132                           'fstatfs',
4133                           'fstatfs64',
4134                           'fsync',
4135                           'ftruncate',
4136                           'futimesat',
4137                           'getdents',
4138                           'getdents64',
4139                           'getpeername',
4140                           'getsockname',
4141                           'getsockopt',
4142                           'inotify_add_watch',
4143                           'inotify_rm_watch',
4144                           'ioctl',
4145                           'listen',
4146                           'llseek',
4147                           'lseek',
4148                           'memfd_create',
4149                           'mmap_pgoff',
4150                           'mq_getsetattr',
4151                           'mq_notify',
```



```
4152                           'mq_open',
4153                           'mq_timedreceive',
4154                           'mq_timedsend',
4155                           'old_readdir',
4156                           'open',
4157                           'open_by_handle_at',
4158                           'openat',
4159                           'perf_event_open',
4160                           'pipe2',
4161                           'pread64',
4162                           'preadv',
4163                           'preadv2',
4164                           'preadv64',
4165                           'preadv64v2',
4166                           'pwrite64',
4167                           'pwritev',
4168                           'pwritev2',
4169                           'pwritev64',
4170                           'pwritev64v2',
4171                           'read',
4172                           'readahead',
4173                           'readv',
4174                           'recvfrom',
4175                           'remap_file_pages',
4176                           'sendfile',
4177                           'sendfile64',
4178                           'sendto',
4179                           'setns',
4180                           'setsockopt',
4181                           'shmat',
4182                           'shmctl',
4183                           'shmdt',
4184                           'shutdown',
4185                           'signalfd4',
4186                           'socket',
4187                           'socketpair',
4188                           'splice',
4189                           'swapoff',
4190                           'swapon',
4191                           'sync_file_range',
4192                           'syncfs',
4193                           'tee',
4194                           'uselib',
4195                           'utime',
4196                           'utimensat',
4197                           'vmsplice',
4198                           'write',
4199                           'writev']),
4200  'io_cancel': set(['brk',
4201                    'get_mempolicy',
4202                    'getrusage',
4203                    'io_cancel',
4204                    'io_destroy',
4205                    'io_getevents',
4206                    'io_setup',
4207                    'io_submit',
4208                    'mbind',
4209                    'migrate_pages',
4210                    'mincore',
4211                    'modify_ldt',
4212                    'move_pages',
4213                    'mremap',
4214                    'prctl',
4215                    'remap_file_pages',
4216                    'shmdt',
4217                    'swapoff']),
```

```
4218  'io_destroy': set(['brk',
4219                     'get_mempolicy',
4220                     'getrusage',
4221                     'io_cancel',
4222                     'io_destroy',
4223                     'io_getevents',
4224                     'io_setup',
4225                     'io_submit',
4226                     'mbind',
4227                     'migrate_pages',
4228                     'mincore',
4229                     'modify_ldt',
4230                     'move_pages',
4231                     'mremap',
4232                     'prctl',
4233                     'remap_file_pages',
4234                     'shmdt',
4235                     'swapoff']),
4236  'io_getevents': set(['brk',
4237                       'capget',
4238                       'clone',
4239                       'fork',
4240                       'get_mempolicy',
4241                       'get_robust_list',
4242                       'getitimer',
4243                       'getpgid',
4244                       'getppid',
4245                       'getpriority',
4246                       'getrusage',
4247                       'getsid',
4248                       'io_cancel',
4249                       'io_destroy',
4250                       'io_getevents',
4251                       'io_setup',
4252                       'io_submit',
4253                       'ioprio_get',
4254                       'ioprio_set',
4255                       'keyctl',
4256                       'kill',
4257                       'mbind',
4258                       'migrate_pages',
4259                       'mincore',
4260                       'modify_ldt',
4261                       'move_pages',
4262                       'mq_timedreceive',
4263                       'mq_timedsend',
4264                       'mremap',
4265                       'msgrcv',
4266                       'perf_event_open',
4267                       'prctl',
4268                       'prlimit64',
4269                       'ptrace',
4270                       'remap_file_pages',
4271                       'rt_sigaction',
4272                       'rt_sigprocmask',
4273                       'rt_sigtimedwait',
4274                       'sched_getaffinity',
4275                       'sched_getattr',
4276                       'sched_getparam',
4277                       'sched_getscheduler',
4278                       'sched_rr_get_interval',
4279                       'sched_setaffinity',
4280                       'sched_setattr',
4281                       'sched_setparam',
4282                       'sched_setscheduler',
4283                       'semtimedop',
```



```
4284                       'setitimer',
4285                       'setns',
4286                       'setpgid',
4287                       'setpriority',
4288                       'setsid',
4289                       'shmdt',
4290                       'sigaction',
4291                       'sigaltstack',
4292                       'signal',
4293                       'swapoff',
4294                       'umount',
4295                       'vfork']),
4296  'io_setup': set(['io_cancel',
4297                   'io_destroy',
4298                   'io_getevents',
4299                   'io_setup',
4300                   'io_submit']),
4301  'io_submit': set(['capget',
4302                    'clone',
4303                    'fork',
4304                    'get_robust_list',
4305                    'getitimer',
4306                    'getpgid',
4307                    'getppid',
4308                    'getpriority',
4309                    'getrusage',
4310                    'getsid',
4311                    'ioperm',
4312                    'ioprio_get',
4313                    'ioprio_set',
4314                    'keyctl',
4315                    'kill',
4316                    'migrate_pages',
4317                    'move_pages',
4318                    'mq_timedreceive',
4319                    'mq_timedsend',
4320                    'msgrcv',
4321                    'perf_event_open',
4322                    'prctl',
4323                    'prlimit64',
4324                    'ptrace',
4325                    'rt_sigaction',
4326                    'rt_sigprocmask',
4327                    'rt_sigtimedwait',
4328                    'sched_getaffinity',
4329                    'sched_getattr',
4330                    'sched_getparam',
4331                    'sched_getscheduler',
4332                    'sched_rr_get_interval',
4333                    'sched_setaffinity',
4334                    'sched_setattr',
4335                    'sched_setparam',
4336                    'sched_setscheduler',
4337                    'semtimedop',
4338                    'setitimer',
4339                    'setns',
4340                    'setpgid',
4341                    'setpriority',
4342                    'setsid',
4343                    'sigaction',
4344                    'sigaltstack',
4345                    'signal',
4346                    'umount',
4347                    'vfork']),
4348  'ioctl': set(['accept4',
4349                'bind',
```

```
4350                'bpf',
4351                'connect',
4352                'copy_file_range',
4353                'epoll_ctl',
4354                'epoll_wait',
4355                'fallocate',
4356                'fchdir',
4357                'fchmod',
4358                'fchown',
4359                'fcntl',
4360                'fcntl64',
4361                'fdatasync',
4362                'fgetxattr',
4363                'flistxattr',
4364                'flock',
4365                'fremovexattr',
4366                'fsetxattr',
4367                'fstatfs',
4368                'fstatfs64',
4369                'fsync',
4370                'ftruncate',
4371                'futimesat',
4372                'getdents',
4373                'getdents64',
4374                'getpeername',
4375                'getsockname',
4376                'getsockopt',
4377                'inotify_add_watch',
4378                'inotify_rm_watch',
4379                'ioctl',
4380                'listen',
4381                'llseek',
4382                'lseek',
4383                'mq_getsetattr',
4384                'mq_notify',
4385                'mq_timedreceive',
4386                'mq_timedsend',
4387                'old_readdir',
4388                'perf_event_open',
4389                'pread64',
4390                'preadv',
4391                'preadv2',
4392                'preadv64',
4393                'preadv64v2',
4394                'pwrite64',
4395                'pwritev',
4396                'pwritev2',
4397                'pwritev64',
4398                'pwritev64v2',
4399                'read',
4400                'readahead',
4401                'readv',
4402                'recvfrom',
4403                'sendfile',
4404                'sendfile64',
4405                'sendto',
4406                'setsockopt',
4407                'shutdown',
4408                'signalfd4',
4409                'splice',
4410                'sync_file_range',
4411                'syncfs',
4412                'tee',
4413                'utime',
4414                'utimensat',
4415                'vmsplice',
```



```
4416                'write',
4417                'writev']),
4418  'ioperm': set(['capget',
4419                 'clone',
4420                 'fork',
4421                 'get_robust_list',
4422                 'getitimer',
4423                 'getpgid',
4424                 'getppid',
4425                 'getpriority',
4426                 'getrusage',
4427                 'getsid',
4428                 'ioperm',
4429                 'ioprio_get',
4430                 'ioprio_set',
4431                 'keyctl',
4432                 'kill',
4433                 'migrate_pages',
4434                 'move_pages',
4435                 'mq_timedreceive',
4436                 'mq_timedsend',
4437                 'msgrcv',
4438                 'perf_event_open',
4439                 'prctl',
4440                 'prlimit64',
4441                 'ptrace',
4442                 'rt_sigaction',
4443                 'rt_sigprocmask',
4444                 'rt_sigtimedwait',
4445                 'sched_getaffinity',
4446                 'sched_getattr',
4447                 'sched_getparam',
4448                 'sched_getscheduler',
4449                 'sched_rr_get_interval',
4450                 'sched_setaffinity',
4451                 'sched_setattr',
4452                 'sched_setparam',
4453                 'sched_setscheduler',
4454                 'semtimedop',
4455                 'setitimer',
4456                 'setns',
4457                 'setpgid',
4458                 'setpriority',
4459                 'setsid',
4460                 'sigaction',
4461                 'sigaltstack',
4462                 'signal',
4463                 'umount',
4464                 'vfork']),
4465  'ioprio_get': set(['capget',
4466                     'clone',
4467                     'fork',
4468                     'get_robust_list',
4469                     'getitimer',
4470                     'getpgid',
4471                     'getppid',
4472                     'getpriority',
4473                     'getrusage',
4474                     'getsid',
4475                     'ioprio_get',
4476                     'ioprio_set',
4477                     'keyctl',
4478                     'kill',
4479                     'migrate_pages',
4480                     'move_pages',
4481                     'mq_timedreceive',
```

```
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'ioprio_set': set(['capget',
                    'clone',
                    'fork',
                    'get_robust_list',
                    'getitimer',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
```



```
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'keyctl': set(['capget',
                'capset',
                'clone',
                'epoll_create1',
                'faccessat',
                'fork',
                'get_robust_list',
                'getgroups',
                'getgroups16',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getresgid',
                'getresgid16',
                'getresuid',
                'getresuid16',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'request_key',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setfsgid',
                'setfsuid',
                'setgid',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setregid',
                'setresgid',
                'setresuid',
                'setreuid',
```

```
4614                 'setsid',
4615                 'setuid',
4616                 'sigaction',
4617                 'sigaltstack',
4618                 'signal',
4619                 'umount',
4620                 'unshare',
4621                 'vfork']),
4622  'kill': set(['capget',
4623               'clone',
4624               'fork',
4625               'get_robust_list',
4626               'getitimer',
4627               'getpgid',
4628               'getppid',
4629               'getpriority',
4630               'getrusage',
4631               'getsid',
4632               'ioprio_get',
4633               'ioprio_set',
4634               'keyctl',
4635               'kill',
4636               'migrate_pages',
4637               'move_pages',
4638               'mq_timedreceive',
4639               'mq_timedsend',
4640               'msgrcv',
4641               'perf_event_open',
4642               'prctl',
4643               'prlimit64',
4644               'ptrace',
4645               'rt_sigaction',
4646               'rt_sigprocmask',
4647               'rt_sigtimedwait',
4648               'sched_getaffinity',
4649               'sched_getattr',
4650               'sched_getparam',
4651               'sched_getscheduler',
4652               'sched_rr_get_interval',
4653               'sched_setaffinity',
4654               'sched_setattr',
4655               'sched_setparam',
4656               'sched_setscheduler',
4657               'semtimedop',
4658               'setitimer',
4659               'setns',
4660               'setpgid',
4661               'setpriority',
4662               'setsid',
4663               'sigaction',
4664               'sigaltstack',
4665               'signal',
4666               'umount',
4667               'vfork']),
4668  'lgetxattr': set(['accept4',
4669                    'acct',
4670                    'dup',
4671                    'dup3',
4672                    'epoll_create1',
4673                    'epoll_ctl',
4674                    'eventfd2',
4675                    'flock',
4676                    'getcwd',
4677                    'lookup_dcookie',
4678                    'memfd_create',
4679                    'mmap_pgoff',
```



```
4680                    'mq_open',
4681                    'open',
4682                    'open_by_handle_at',
4683                    'openat',
4684                    'perf_event_open',
4685                    'pipe2',
4686                    'pivot_root',
4687                    'quotactl',
4688                    'remap_file_pages',
4689                    'setns',
4690                    'shmat',
4691                    'shmctl',
4692                    'shmdt',
4693                    'socket',
4694                    'socketpair',
4695                    'swapoff',
4696                    'swapon',
4697                    'unshare',
4698                    'uselib']),
4699  'linkat': set(['accept4',
4700                 'acct',
4701                 'dup',
4702                 'dup3',
4703                 'epoll_create1',
4704                 'epoll_ctl',
4705                 'eventfd2',
4706                 'flock',
4707                 'getcwd',
4708                 'lookup_dcookie',
4709                 'memfd_create',
4710                 'mmap_pgoff',
4711                 'mq_open',
4712                 'open',
4713                 'open_by_handle_at',
4714                 'openat',
4715                 'perf_event_open',
4716                 'pipe2',
4717                 'pivot_root',
4718                 'quotactl',
4719                 'remap_file_pages',
4720                 'setns',
4721                 'shmat',
4722                 'shmctl',
4723                 'shmdt',
4724                 'socket',
4725                 'socketpair',
4726                 'swapoff',
4727                 'swapon',
4728                 'unshare',
4729                 'uselib']),
4730  'listen': set(['accept4',
4731                 'bind',
4732                 'bpf',
4733                 'connect',
4734                 'copy_file_range',
4735                 'epoll_ctl',
4736                 'epoll_wait',
4737                 'fallocate',
4738                 'fchdir',
4739                 'fchmod',
4740                 'fchown',
4741                 'fcntl',
4742                 'fcntl64',
4743                 'fdatasync',
4744                 'fgetxattr',
4745                 'flistxattr',
```

```
4746                 'flock',
4747                 'fremovexattr',
4748                 'fsetxattr',
4749                 'fstatfs',
4750                 'fstatfs64',
4751                 'fsync',
4752                 'ftruncate',
4753                 'futimesat',
4754                 'getdents',
4755                 'getdents64',
4756                 'getpeername',
4757                 'getsockname',
4758                 'getsockopt',
4759                 'inotify_add_watch',
4760                 'inotify_rm_watch',
4761                 'ioctl',
4762                 'listen',
4763                 'llseek',
4764                 'lseek',
4765                 'mq_getsetattr',
4766                 'mq_notify',
4767                 'mq_timedreceive',
4768                 'mq_timedsend',
4769                 'old_readdir',
4770                 'perf_event_open',
4771                 'pread64',
4772                 'preadv',
4773                 'preadv2',
4774                 'preadv64',
4775                 'preadv64v2',
4776                 'pwrite64',
4777                 'pwritev',
4778                 'pwritev2',
4779                 'pwritev64',
4780                 'pwritev64v2',
4781                 'read',
4782                 'readahead',
4783                 'readv',
4784                 'recvfrom',
4785                 'sendfile',
4786                 'sendfile64',
4787                 'sendto',
4788                 'setsockopt',
4789                 'shutdown',
4790                 'signalfd4',
4791                 'splice',
4792                 'sync_file_range',
4793                 'syncfs',
4794                 'tee',
4795                 'utime',
4796                 'utimensat',
4797                 'vmsplice',
4798                 'write',
4799                 'writev']),
4800  'listxattr': set(['accept4',
4801                    'acct',
4802                    'dup',
4803                    'dup3',
4804                    'epoll_create1',
4805                    'epoll_ctl',
4806                    'eventfd2',
4807                    'flock',
4808                    'getcwd',
4809                    'lookup_dcookie',
4810                    'memfd_create',
4811                    'mmap_pgoff',
```



```
4812                    'mq_open',
4813                    'open',
4814                    'open_by_handle_at',
4815                    'openat',
4816                    'perf_event_open',
4817                    'pipe2',
4818                    'pivot_root',
4819                    'quotactl',
4820                    'remap_file_pages',
4821                    'setns',
4822                    'shmat',
4823                    'shmctl',
4824                    'shmdt',
4825                    'socket',
4826                    'socketpair',
4827                    'swapoff',
4828                    'swapon',
4829                    'unshare',
4830                    'uselib']),
4831  'llistxattr': set(['accept4',
4832                     'acct',
4833                     'dup',
4834                     'dup3',
4835                     'epoll_create1',
4836                     'epoll_ctl',
4837                     'eventfd2',
4838                     'flock',
4839                     'getcwd',
4840                     'lookup_dcookie',
4841                     'memfd_create',
4842                     'mmap_pgoff',
4843                     'mq_open',
4844                     'open',
4845                     'open_by_handle_at',
4846                     'openat',
4847                     'perf_event_open',
4848                     'pipe2',
4849                     'pivot_root',
4850                     'quotactl',
4851                     'remap_file_pages',
4852                     'setns',
4853                     'shmat',
4854                     'shmctl',
4855                     'shmdt',
4856                     'socket',
4857                     'socketpair',
4858                     'swapoff',
4859                     'swapon',
4860                     'unshare',
4861                     'uselib']),
4862  'llseek': set(['accept4',
4863                 'bind',
4864                 'bpf',
4865                 'connect',
4866                 'copy_file_range',
4867                 'epoll_ctl',
4868                 'epoll_wait',
4869                 'fallocate',
4870                 'fchdir',
4871                 'fchmod',
4872                 'fchown',
4873                 'fcntl',
4874                 'fcntl64',
4875                 'fdatasync',
4876                 'fgetxattr',
4877                 'flistxattr',
```

```
4878                 'flock',
4879                 'fremovexattr',
4880                 'fsetxattr',
4881                 'fstatfs',
4882                 'fstatfs64',
4883                 'fsync',
4884                 'ftruncate',
4885                 'futimesat',
4886                 'getdents',
4887                 'getdents64',
4888                 'getpeername',
4889                 'getsockname',
4890                 'getsockopt',
4891                 'inotify_add_watch',
4892                 'inotify_rm_watch',
4893                 'ioctl',
4894                 'listen',
4895                 'llseek',
4896                 'lseek',
4897                 'mq_getsetattr',
4898                 'mq_notify',
4899                 'mq_timedreceive',
4900                 'mq_timedsend',
4901                 'old_readdir',
4902                 'perf_event_open',
4903                 'pread64',
4904                 'preadv',
4905                 'preadv2',
4906                 'preadv64',
4907                 'preadv64v2',
4908                 'pwrite64',
4909                 'pwritev',
4910                 'pwritev2',
4911                 'pwritev64',
4912                 'pwritev64v2',
4913                 'read',
4914                 'readahead',
4915                 'readv',
4916                 'recvfrom',
4917                 'sendfile',
4918                 'sendfile64',
4919                 'sendto',
4920                 'setsockopt',
4921                 'shutdown',
4922                 'signalfd4',
4923                 'splice',
4924                 'sync_file_range',
4925                 'syncfs',
4926                 'tee',
4927                 'utime',
4928                 'utimensat',
4929                 'vmsplice',
4930                 'write',
4931                 'writev']),
4932  'lremovexattr': set(['accept4',
4933                       'acct',
4934                       'dup',
4935                       'dup3',
4936                       'epoll_create1',
4937                       'epoll_ctl',
4938                       'eventfd2',
4939                       'flock',
4940                       'getcwd',
4941                       'lookup_dcookie',
4942                       'memfd_create',
4943                       'mmap_pgoff',
```



```
4944                       'mq_open',
4945                       'open',
4946                       'open_by_handle_at',
4947                       'openat',
4948                       'perf_event_open',
4949                       'pipe2',
4950                       'pivot_root',
4951                       'quotactl',
4952                       'remap_file_pages',
4953                       'setns',
4954                       'shmat',
4955                       'shmctl',
4956                       'shmdt',
4957                       'socket',
4958                       'socketpair',
4959                       'swapoff',
4960                       'swapon',
4961                       'unshare',
4962                       'uselib']),
4963  'lseek': set(['accept4',
4964                'bind',
4965                'bpf',
4966                'connect',
4967                'copy_file_range',
4968                'epoll_ctl',
4969                'epoll_wait',
4970                'fallocate',
4971                'fchdir',
4972                'fchmod',
4973                'fchown',
4974                'fcntl',
4975                'fcntl64',
4976                'fdatasync',
4977                'fgetxattr',
4978                'flistxattr',
4979                'flock',
4980                'fremovexattr',
4981                'fsetxattr',
4982                'fstatfs',
4983                'fstatfs64',
4984                'fsync',
4985                'ftruncate',
4986                'futimesat',
4987                'getdents',
4988                'getdents64',
4989                'getpeername',
4990                'getsockname',
4991                'getsockopt',
4992                'inotify_add_watch',
4993                'inotify_rm_watch',
4994                'ioctl',
4995                'listen',
4996                'llseek',
4997                'lseek',
4998                'mq_getsetattr',
4999                'mq_notify',
5000                'mq_timedreceive',
5001                'mq_timedsend',
5002                'old_readdir',
5003                'perf_event_open',
5004                'pread64',
5005                'preadv',
5006                'preadv2',
5007                'preadv64',
5008                'preadv64v2',
5009                'pwrite64',
```

```
5010                 'pwritev',
5011                 'pwritev2',
5012                 'pwritev64',
5013                 'pwritev64v2',
5014                 'read',
5015                 'readahead',
5016                 'readv',
5017                 'recvfrom',
5018                 'sendfile',
5019                 'sendfile64',
5020                 'sendto',
5021                 'setsockopt',
5022                 'shutdown',
5023                 'signalfd4',
5024                 'splice',
5025                 'sync_file_range',
5026                 'syncfs',
5027                 'tee',
5028                 'utime',
5029                 'utimensat',
5030                 'vmsplice',
5031                 'write',
5032                 'writev']),
5033  'lsetxattr': set(['accept4',
5034                    'acct',
5035                    'dup',
5036                    'dup3',
5037                    'epoll_create1',
5038                    'epoll_ctl',
5039                    'eventfd2',
5040                    'flock',
5041                    'getcwd',
5042                    'lookup_dcookie',
5043                    'memfd_create',
5044                    'mmap_pgoff',
5045                    'mq_open',
5046                    'open',
5047                    'open_by_handle_at',
5048                    'openat',
5049                    'perf_event_open',
5050                    'pipe2',
5051                    'pivot_root',
5052                    'quotactl',
5053                    'remap_file_pages',
5054                    'setns',
5055                    'shmat',
5056                    'shmctl',
5057                    'shmdt',
5058                    'socket',
5059                    'socketpair',
5060                    'swapoff',
5061                    'swapon',
5062                    'unshare',
5063                    'uselib']),
5064  'lstat': set(['fstat', 'lstat', 'newfstat', 'stat']),
5065  'madvise': set(['brk',
5066                  'get_mempolicy',
5067                  'madvise',
5068                  'mincore',
5069                  'mlockall',
5070                  'mprotect',
5071                  'mremap',
5072                  'munlock',
5073                  'munlockall',
5074                  'pkey_mprotect',
5075                  'prctl',
```



```
5076                 'remap_file_pages',
5077                 'shmdt']),
5078  'migrate_pages': set(['capget',
5079                        'clone',
5080                        'fork',
5081                        'get_robust_list',
5082                        'getitimer',
5083                        'getpgid',
5084                        'getppid',
5085                        'getpriority',
5086                        'getrusage',
5087                        'getsid',
5088                        'ioperm',
5089                        'ioprio_get',
5090                        'ioprio_set',
5091                        'keyctl',
5092                        'kill',
5093                        'migrate_pages',
5094                        'move_pages',
5095                        'mq_timedreceive',
5096                        'mq_timedsend',
5097                        'msgrcv',
5098                        'perf_event_open',
5099                        'prctl',
5100                        'prlimit64',
5101                        'ptrace',
5102                        'rt_sigaction',
5103                        'rt_sigprocmask',
5104                        'rt_sigtimedwait',
5105                        'sched_getaffinity',
5106                        'sched_getattr',
5107                        'sched_getparam',
5108                        'sched_getscheduler',
5109                        'sched_rr_get_interval',
5110                        'sched_setaffinity',
5111                        'sched_setattr',
5112                        'sched_setparam',
5113                        'sched_setscheduler',
5114                        'semtimedop',
5115                        'setitimer',
5116                        'setns',
5117                        'setpgid',
5118                        'setpriority',
5119                        'setresuid',
5120                        'setreuid',
5121                        'setsid',
5122                        'setuid',
5123                        'sigaction',
5124                        'sigaltstack',
5125                        'signal',
5126                        'umount',
5127                        'vfork']),
5128  'mincore': set(['brk',
5129                  'capget',
5130                  'clone',
5131                  'fork',
5132                  'get_mempolicy',
5133                  'get_robust_list',
5134                  'getitimer',
5135                  'getpgid',
5136                  'getppid',
5137                  'getpriority',
5138                  'getrusage',
5139                  'getsid',
5140                  'ioperm',
5141                  'ioprio_get',
```

```
                'ioprio_set',
                'keyctl',
                'kill',
                'madvise',
                'migrate_pages',
                'mincore',
                'mlockall',
                'move_pages',
                'mprotect',
                'mq_timedreceive',
                'mq_timedsend',
                'mremap',
                'msgrcv',
                'munlock',
                'munlockall',
                'perf_event_open',
                'pkey_mprotect',
                'prctl',
                'prlimit64',
                'ptrace',
                'remap_file_pages',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmdt',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
'mkdirat': set(['quotactl', 'swapon', 'syncfs', 'umount', 'ustat']),
'mknodat': set(['quotactl', 'swapon', 'syncfs', 'umount', 'ustat']),
'mlock': set(['capget',
              'clone',
              'fork',
              'get_robust_list',
              'getitimer',
              'getpgid',
              'getppid',
              'getpriority',
              'getrusage',
              'getsid',
              'ioprio_get',
              'ioprio_set',
              'keyctl',
              'kill',
              'migrate_pages',
              'move_pages',
              'mq_timedreceive',
              'mq_timedsend',
              'msgrcv',
```



```
              'perf_event_open',
              'prctl',
              'prlimit64',
              'ptrace',
              'rt_sigaction',
              'rt_sigprocmask',
              'rt_sigtimedwait',
              'sched_getaffinity',
              'sched_getattr',
              'sched_getparam',
              'sched_getscheduler',
              'sched_rr_get_interval',
              'sched_setaffinity',
              'sched_setattr',
              'sched_setparam',
              'sched_setscheduler',
              'semtimedop',
              'setitimer',
              'setns',
              'setpgid',
              'setpriority',
              'setsid',
              'sigaction',
              'sigaltstack',
              'signal',
              'umount',
              'vfork']),
'mlock2': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'migrate_pages',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'msgrcv',
               'perf_event_open',
               'prctl',
               'prlimit64',
               'ptrace',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
               'semtimedop',
               'setitimer',
               'setns',
               'setpgid',
```

```
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'mlockall': set(['brk',
                  'capget',
                  'clone',
                  'fork',
                  'get_mempolicy',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'io_cancel',
                  'io_destroy',
                  'io_getevents',
                  'io_setup',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'madvise',
                  'mbind',
                  'migrate_pages',
                  'mincore',
                  'mlockall',
                  'modify_ldt',
                  'move_pages',
                  'mprotect',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mremap',
                  'msgrcv',
                  'munlock',
                  'munlockall',
                  'perf_event_open',
                  'personality',
                  'pkey_mprotect',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'remap_file_pages',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
```



```
                  'shmdt',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'swapoff',
                  'umount',
                  'vfork']),
 'mmap_pgoff': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_open',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'remap_file_pages',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'uselib']),
 'modify_ldt': set(['brk',
                    'get_mempolicy',
                    'get_thread_area',
                    'getrusage',
                    'io_cancel',
                    'io_destroy',
                    'io_getevents',
                    'io_setup',
                    'mbind',
                    'migrate_pages',
                    'mincore',
                    'modify_ldt',
                    'move_pages',
                    'mremap',
                    'prctl',
                    'remap_file_pages',
                    'set_thread_area',
                    'shmdt',
                    'swapoff']),
 'mount': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
```

```
5406                 'migrate_pages',
5407                 'mount',
5408                 'move_pages',
5409                 'mq_timedreceive',
5410                 'mq_timedsend',
5411                 'msgrcv',
5412                 'perf_event_open',
5413                 'personality',
5414                 'prctl',
5415                 'prlimit64',
5416                 'ptrace',
5417                 'rt_sigaction',
5418                 'rt_sigprocmask',
5419                 'rt_sigtimedwait',
5420                 'sched_getaffinity',
5421                 'sched_getattr',
5422                 'sched_getparam',
5423                 'sched_getscheduler',
5424                 'sched_rr_get_interval',
5425                 'sched_setaffinity',
5426                 'sched_setattr',
5427                 'sched_setparam',
5428                 'sched_setscheduler',
5429                 'semtimedop',
5430                 'setitimer',
5431                 'setns',
5432                 'setpgid',
5433                 'setpriority',
5434                 'setsid',
5435                 'sigaction',
5436                 'sigaltstack',
5437                 'signal',
5438                 'umount',
5439                 'vfork']),
5440  'mprotect': set(['brk',
5441                   'capget',
5442                   'clone',
5443                   'fork',
5444                   'get_mempolicy',
5445                   'get_robust_list',
5446                   'getitimer',
5447                   'getpgid',
5448                   'getppid',
5449                   'getpriority',
5450                   'getrusage',
5451                   'getsid',
5452                   'ioprio_get',
5453                   'ioprio_set',
5454                   'keyctl',
5455                   'kill',
5456                   'madvise',
5457                   'migrate_pages',
5458                   'mincore',
5459                   'mlockall',
5460                   'move_pages',
5461                   'mprotect',
5462                   'mq_timedreceive',
5463                   'mq_timedsend',
5464                   'mremap',
5465                   'msgrcv',
5466                   'munlock',
5467                   'munlockall',
5468                   'perf_event_open',
5469                   'personality',
5470                   'pkey_mprotect',
5471                   'prctl',
```



```
5472                   'prlimit64',
5473                   'ptrace',
5474                   'remap_file_pages',
5475                   'rt_sigaction',
5476                   'rt_sigprocmask',
5477                   'rt_sigtimedwait',
5478                   'sched_getaffinity',
5479                   'sched_getattr',
5480                   'sched_getparam',
5481                   'sched_getscheduler',
5482                   'sched_rr_get_interval',
5483                   'sched_setaffinity',
5484                   'sched_setattr',
5485                   'sched_setparam',
5486                   'sched_setscheduler',
5487                   'semtimedop',
5488                   'setitimer',
5489                   'setns',
5490                   'setpgid',
5491                   'setpriority',
5492                   'setsid',
5493                   'shmdt',
5494                   'sigaction',
5495                   'sigaltstack',
5496                   'signal',
5497                   'umount',
5498                   'vfork']),
5499  'mq_getsetattr': set(['accept4',
5500                        'acct',
5501                        'bind',
5502                        'bpf',
5503                        'connect',
5504                        'copy_file_range',
5505                        'dup',
5506                        'dup3',
5507                        'epoll_create1',
5508                        'epoll_ctl',
5509                        'epoll_wait',
5510                        'eventfd2',
5511                        'fallocate',
5512                        'fchdir',
5513                        'fchmod',
5514                        'fchown',
5515                        'fcntl',
5516                        'fcntl64',
5517                        'fdatasync',
5518                        'fgetxattr',
5519                        'flistxattr',
5520                        'flock',
5521                        'fremovexattr',
5522                        'fsetxattr',
5523                        'fstatfs',
5524                        'fstatfs64',
5525                        'fsync',
5526                        'ftruncate',
5527                        'futimesat',
5528                        'getdents',
5529                        'getdents64',
5530                        'getpeername',
5531                        'getsockname',
5532                        'getsockopt',
5533                        'inotify_add_watch',
5534                        'inotify_rm_watch',
5535                        'ioctl',
5536                        'listen',
5537                        'llseek',
```

```
                      'lseek',
                      'memfd_create',
                      'mmap_pgoff',
                      'mq_getsetattr',
                      'mq_notify',
                      'mq_open',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'old_readdir',
                      'open',
                      'open_by_handle_at',
                      'openat',
                      'perf_event_open',
                      'pipe2',
                      'pread64',
                      'preadv',
                      'preadv2',
                      'preadv64',
                      'preadv64v2',
                      'pwrite64',
                      'pwritev',
                      'pwritev2',
                      'pwritev64',
                      'pwritev64v2',
                      'read',
                      'readahead',
                      'readv',
                      'recvfrom',
                      'remap_file_pages',
                      'sendfile',
                      'sendfile64',
                      'sendto',
                      'setns',
                      'setsockopt',
                      'shmat',
                      'shmctl',
                      'shmdt',
                      'shutdown',
                      'signalfd4',
                      'socket',
                      'socketpair',
                      'splice',
                      'swapoff',
                      'swapon',
                      'sync_file_range',
                      'syncfs',
                      'tee',
                      'uselib',
                      'utime',
                      'utimensat',
                      'vmsplice',
                      'write',
                      'writev']),
  'mq_notify': set(['accept4',
                    'acct',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'epoll_wait',
                    'eventfd2',
                    'fallocate',
```



```
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'rt_sigqueueinfo',
                    'rt_sigreturn',
                    'rt_sigtimedwait',
                    'rt_tgsigqueueinfo',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setns',
                    'setsockopt',
                    'shmat',
                    'shmctl',
```

```
                    'shmdt',
                    'shutdown',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'tgkill',
                    'timer_create',
                    'tkill',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'mq_open': set(['accept4',
                 'acct',
                 'dup',
                 'dup3',
                 'epoll_create1',
                 'epoll_ctl',
                 'eventfd2',
                 'flock',
                 'getcwd',
                 'lookup_dcookie',
                 'memfd_create',
                 'mmap_pgoff',
                 'mq_open',
                 'mq_unlink',
                 'open',
                 'open_by_handle_at',
                 'openat',
                 'perf_event_open',
                 'pipe2',
                 'pivot_root',
                 'quotactl',
                 'remap_file_pages',
                 'renameat2',
                 'rmdir',
                 'setns',
                 'shmat',
                 'shmctl',
                 'shmdt',
                 'socket',
                 'socketpair',
                 'swapoff',
                 'swapon',
                 'symlinkat',
                 'sysfs',
                 'unlink',
                 'unlinkat',
                 'unshare',
                 'uselib']),
 'mq_timedreceive': set(['accept4',
                         'acct',
                         'bind',
                         'bpf',
                         'connect',
                         'copy_file_range',
                         'dup',
                         'dup3',
```



```
                         'epoll_create1',
                         'epoll_ctl',
                         'epoll_wait',
                         'eventfd2',
                         'fallocate',
                         'fchdir',
                         'fchmod',
                         'fchown',
                         'fcntl',
                         'fcntl64',
                         'fdatasync',
                         'fgetxattr',
                         'flistxattr',
                         'flock',
                         'fremovexattr',
                         'fsetxattr',
                         'fstatfs',
                         'fstatfs64',
                         'fsync',
                         'ftruncate',
                         'futimesat',
                         'getdents',
                         'getdents64',
                         'getpeername',
                         'getsockname',
                         'getsockopt',
                         'inotify_add_watch',
                         'inotify_rm_watch',
                         'ioctl',
                         'listen',
                         'llseek',
                         'lseek',
                         'memfd_create',
                         'mmap_pgoff',
                         'mq_getsetattr',
                         'mq_notify',
                         'mq_open',
                         'mq_timedreceive',
                         'mq_timedsend',
                         'msgrcv',
                         'msgsnd',
                         'old_readdir',
                         'open',
                         'open_by_handle_at',
                         'openat',
                         'perf_event_open',
                         'pipe2',
                         'pread64',
                         'preadv',
                         'preadv2',
                         'preadv64',
                         'preadv64v2',
                         'pwrite64',
                         'pwritev',
                         'pwritev2',
                         'pwritev64',
                         'pwritev64v2',
                         'read',
                         'readahead',
                         'readv',
                         'recvfrom',
                         'remap_file_pages',
                         'sendfile',
                         'sendfile64',
                         'sendto',
                         'setns',
```

```
                        'setsockopt',
                        'shmat',
                        'shmctl',
                        'shmdt',
                        'shutdown',
                        'signalfd4',
                        'socket',
                        'socketpair',
                        'splice',
                        'swapoff',
                        'swapon',
                        'sync_file_range',
                        'syncfs',
                        'tee',
                        'uselib',
                        'utime',
                        'utimensat',
                        'vmsplice',
                        'write',
                        'writev']),
 'mq_timedsend': set(['accept4',
                      'acct',
                      'bind',
                      'bpf',
                      'connect',
                      'copy_file_range',
                      'dup',
                      'dup3',
                      'epoll_create1',
                      'epoll_ctl',
                      'epoll_wait',
                      'eventfd2',
                      'fallocate',
                      'fchdir',
                      'fchmod',
                      'fchown',
                      'fcntl',
                      'fcntl64',
                      'fdatasync',
                      'fgetxattr',
                      'flistxattr',
                      'flock',
                      'fremovexattr',
                      'fsetxattr',
                      'fstatfs',
                      'fstatfs64',
                      'fsync',
                      'ftruncate',
                      'futimesat',
                      'getdents',
                      'getdents64',
                      'getpeername',
                      'getsockname',
                      'getsockopt',
                      'inotify_add_watch',
                      'inotify_rm_watch',
                      'ioctl',
                      'listen',
                      'llseek',
                      'lseek',
                      'memfd_create',
                      'mmap_pgoff',
                      'mq_getsetattr',
                      'mq_notify',
                      'mq_open',
                      'mq_timedreceive',
```



```
                      'mq_timedsend',
                      'old_readdir',
                      'open',
                      'open_by_handle_at',
                      'openat',
                      'perf_event_open',
                      'pipe2',
                      'pread64',
                      'preadv',
                      'preadv2',
                      'preadv64',
                      'preadv64v2',
                      'pwrite64',
                      'pwritev',
                      'pwritev2',
                      'pwritev64',
                      'pwritev64v2',
                      'read',
                      'readahead',
                      'readv',
                      'recvfrom',
                      'remap_file_pages',
                      'sendfile',
                      'sendfile64',
                      'sendto',
                      'setns',
                      'setsockopt',
                      'shmat',
                      'shmctl',
                      'shmdt',
                      'shutdown',
                      'signalfd4',
                      'socket',
                      'socketpair',
                      'splice',
                      'swapoff',
                      'swapon',
                      'sync_file_range',
                      'syncfs',
                      'tee',
                      'uselib',
                      'utime',
                      'utimensat',
                      'vmsplice',
                      'write',
                      'writev']),
 'mremap': set(['brk',
                'capget',
                'clone',
                'fork',
                'get_mempolicy',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'io_cancel',
                'io_destroy',
                'io_getevents',
                'io_setup',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
```

```
                'madvise',
                'mbind',
                'migrate_pages',
                'mincore',
                'mlockall',
                'modify_ldt',
                'move_pages',
                'mprotect',
                'mq_timedreceive',
                'mq_timedsend',
                'mremap',
                'msgrcv',
                'munlock',
                'munlockall',
                'perf_event_open',
                'personality',
                'pkey_mprotect',
                'prctl',
                'prlimit64',
                'ptrace',
                'remap_file_pages',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmdt',
                'sigaction',
                'sigaltstack',
                'signal',
                'swapoff',
                'umount',
                'vfork']),
 'msgctl': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioperm',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
```



```
                'mq_unlink',
                'msgctl',
                'msgget',
                'msgrcv',
                'msgsnd',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semctl',
                'semget',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmat',
                'shmctl',
                'shmget',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'msgrcv': set(['mq_timedreceive', 'mq_timedsend', 'msgrcv', 'msgsnd']),
 'msgsnd': set(['mq_open',
                'mq_unlink',
                'msgctl',
                'msgget',
                'msgrcv',
                'msgsnd',
                'semctl',
                'semget',
                'semtimedop',
                'setns',
                'shmat',
                'shmctl',
                'shmget']),
 'munlock': set(['brk',
                 'capget',
                 'clone',
                 'fork',
                 'get_mempolicy',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
```

```
                  'kill',
                  'madvise',
                  'migrate_pages',
                  'mincore',
                  'mlockall',
                  'move_pages',
                  'mprotect',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mremap',
                  'msgrcv',
                  'munlock',
                  'munlockall',
                  'perf_event_open',
                  'pkey_mprotect',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'remap_file_pages',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'shmdt',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'munlockall': set(['brk',
                    'get_mempolicy',
                    'madvise',
                    'mincore',
                    'mlockall',
                    'mprotect',
                    'mremap',
                    'munlock',
                    'munlockall',
                    'pkey_mprotect',
                    'prctl',
                    'remap_file_pages',
                    'shmdt']),
 'name_to_handle_at': set(['accept4',
                           'acct',
                           'dup',
                           'dup3',
                           'epoll_create1',
                           'epoll_ctl',
                           'eventfd2',
                           'flock',
                           'getcwd',
                           'lookup_dcookie',
```



```
                           'memfd_create',
                           'mmap_pgoff',
                           'mq_open',
                           'name_to_handle_at',
                           'open',
                           'open_by_handle_at',
                           'openat',
                           'perf_event_open',
                           'pipe2',
                           'pivot_root',
                           'quotactl',
                           'remap_file_pages',
                           'setns',
                           'shmat',
                           'shmctl',
                           'shmdt',
                           'socket',
                           'socketpair',
                           'swapoff',
                           'swapon',
                           'syncfs',
                           'umount',
                           'unshare',
                           'uselib',
                           'ustat']),
 'nanosleep': set(['clock_nanosleep',
                   'epoll_wait',
                   'faccessat',
                   'fchmod',
                   'fchmodat',
                   'fchown',
                   'fchownat',
                   'fstat',
                   'ftruncate',
                   'futex',
                   'futimesat',
                   'inotify_add_watch',
                   'io_getevents',
                   'ioctl',
                   'linkat',
                   'memfd_create',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'mq_unlink',
                   'nanosleep',
                   'newfstat',
                   'poll',
                   'ppoll',
                   'pselect6',
                   'readlinkat',
                   'recvmmsg',
                   'rt_sigtimedwait',
                   'sched_rr_get_interval',
                   'select',
                   'semtimedop',
                   'sendfile',
                   'sendfile64',
                   'settimeofday',
                   'stime',
                   'swapoff',
                   'swapon',
                   'timer_gettime',
                   'timer_settime',
                   'timerfd_gettime',
```

```
                  'timerfd_settime',
                  'unlink',
                  'unlinkat',
                  'uselib',
                  'utime']),
 'newfstat': set(['fstat', 'newfstat', 'newfstatat', 'newlstat', 'newstat']),
 'newfstatat': set(['fstat', 'newfstat', 'newfstatat', 'newlstat', 'newstat']),
 'newlstat': set(['fstat', 'newfstat', 'newfstatat', 'newlstat', 'newstat']),
 'newstat': set(['fstat', 'newfstat', 'newfstatat', 'newlstat', 'newstat']),
 'newuname': set(['capget',
                  'clone',
                  'fork',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'personality',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'old_getrlimit': set(['old_getrlimit', 'prlimit64', 'setrlimit']),
 'old_readdir': set(['accept4',
                     'bind',
                     'bpf',
                     'connect',
                     'copy_file_range',
                     'epoll_ctl',
                     'epoll_wait',
                     'fallocate',
                     'fchdir',
```



```
                     'fchmod',
                     'fchown',
                     'fcntl',
                     'fcntl64',
                     'fdatasync',
                     'fgetxattr',
                     'flistxattr',
                     'flock',
                     'fremovexattr',
                     'fsetxattr',
                     'fstatfs',
                     'fstatfs64',
                     'fsync',
                     'ftruncate',
                     'futimesat',
                     'getdents',
                     'getdents64',
                     'getpeername',
                     'getsockname',
                     'getsockopt',
                     'inotify_add_watch',
                     'inotify_rm_watch',
                     'ioctl',
                     'listen',
                     'llseek',
                     'lseek',
                     'mq_getsetattr',
                     'mq_notify',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'old_readdir',
                     'perf_event_open',
                     'pread64',
                     'preadv',
                     'preadv2',
                     'preadv64',
                     'preadv64v2',
                     'pwrite64',
                     'pwritev',
                     'pwritev2',
                     'pwritev64',
                     'pwritev64v2',
                     'read',
                     'readahead',
                     'readv',
                     'recvfrom',
                     'sendfile',
                     'sendfile64',
                     'sendto',
                     'setsockopt',
                     'shutdown',
                     'signalfd4',
                     'splice',
                     'sync_file_range',
                     'syncfs',
                     'tee',
                     'utime',
                     'utimensat',
                     'vmsplice',
                     'write',
                     'writev']),
 'olduname': set(['capget',
                  'clone',
                  'fork',
                  'get_robust_list',
                  'getitimer',
```

```
6330                    'getpgid',
6331                    'getppid',
6332                    'getpriority',
6333                    'getrusage',
6334                    'getsid',
6335                    'ioperm',
6336                    'ioprio_get',
6337                    'ioprio_set',
6338                    'keyctl',
6339                    'kill',
6340                    'migrate_pages',
6341                    'move_pages',
6342                    'mq_timedreceive',
6343                    'mq_timedsend',
6344                    'msgrcv',
6345                    'perf_event_open',
6346                    'personality',
6347                    'prctl',
6348                    'prlimit64',
6349                    'ptrace',
6350                    'rt_sigaction',
6351                    'rt_sigprocmask',
6352                    'rt_sigtimedwait',
6353                    'sched_getaffinity',
6354                    'sched_getattr',
6355                    'sched_getparam',
6356                    'sched_getscheduler',
6357                    'sched_rr_get_interval',
6358                    'sched_setaffinity',
6359                    'sched_setattr',
6360                    'sched_setparam',
6361                    'sched_setscheduler',
6362                    'semtimedop',
6363                    'setitimer',
6364                    'setns',
6365                    'setpgid',
6366                    'setpriority',
6367                    'setsid',
6368                    'sigaction',
6369                    'sigaltstack',
6370                    'signal',
6371                    'umount',
6372                    'vfork']),
6373  'open_by_handle_at': set(['accept4',
6374                            'acct',
6375                            'dup',
6376                            'dup3',
6377                            'epoll_create1',
6378                            'epoll_ctl',
6379                            'eventfd2',
6380                            'flock',
6381                            'getcwd',
6382                            'lookup_dcookie',
6383                            'memfd_create',
6384                            'mmap_pgoff',
6385                            'mq_open',
6386                            'open',
6387                            'open_by_handle_at',
6388                            'openat',
6389                            'perf_event_open',
6390                            'pipe2',
6391                            'pivot_root',
6392                            'quotactl',
6393                            'remap_file_pages',
6394                            'setns',
6395                            'shmat',
```



```
6396                            'shmctl',
6397                            'shmdt',
6398                            'socket',
6399                            'socketpair',
6400                            'swapoff',
6401                            'swapon',
6402                            'unshare',
6403                            'uselib']),
6404  'perf_event_open': set(['accept4',
6405                          'acct',
6406                          'bind',
6407                          'bpf',
6408                          'brk',
6409                          'capget',
6410                          'clone',
6411                          'connect',
6412                          'copy_file_range',
6413                          'delete_module',
6414                          'dup',
6415                          'dup2',
6416                          'dup3',
6417                          'epoll_create1',
6418                          'epoll_ctl',
6419                          'epoll_wait',
6420                          'eventfd2',
6421                          'exit_group',
6422                          'faccessat',
6423                          'fallocate',
6424                          'fchdir',
6425                          'fchmod',
6426                          'fchmodat',
6427                          'fchown',
6428                          'fchownat',
6429                          'fcntl',
6430                          'fcntl64',
6431                          'fdatasync',
6432                          'fgetxattr',
6433                          'finit_module',
6434                          'flistxattr',
6435                          'flock',
6436                          'fork',
6437                          'fremovexattr',
6438                          'fsetxattr',
6439                          'fstatfs',
6440                          'fstatfs64',
6441                          'fsync',
6442                          'ftruncate',
6443                          'futimesat',
6444                          'get_curr_temp',
6445                          'get_mempolicy',
6446                          'get_robust_list',
6447                          'get_trip_temp',
6448                          'getcwd',
6449                          'getdents',
6450                          'getdents64',
6451                          'getgroups',
6452                          'getgroups16',
6453                          'getitimer',
6454                          'getpeername',
6455                          'getpgid',
6456                          'getppid',
6457                          'getpriority',
6458                          'getrusage',
6459                          'getsid',
6460                          'getsockname',
6461                          'getsockopt',
```

```
6462                        'init_module',
6463                        'inotify_add_watch',
6464                        'inotify_init1',
6465                        'inotify_rm_watch',
6466                        'io_cancel',
6467                        'io_destroy',
6468                        'io_getevents',
6469                        'io_setup',
6470                        'io_submit',
6471                        'ioctl',
6472                        'ioperm',
6473                        'ioprio_get',
6474                        'ioprio_set',
6475                        'kexec_load',
6476                        'keyctl',
6477                        'kill',
6478                        'linkat',
6479                        'listen',
6480                        'llseek',
6481                        'lookup_dcookie',
6482                        'lseek',
6483                        'madvise',
6484                        'mbind',
6485                        'memfd_create',
6486                        'migrate_pages',
6487                        'mincore',
6488                        'mkdirat',
6489                        'mknodat',
6490                        'mlockall',
6491                        'mmap_pgoff',
6492                        'modify_ldt',
6493                        'move_pages',
6494                        'mprotect',
6495                        'mq_getsetattr',
6496                        'mq_notify',
6497                        'mq_open',
6498                        'mq_timedreceive',
6499                        'mq_timedsend',
6500                        'mq_unlink',
6501                        'mremap',
6502                        'msgctl',
6503                        'msgrcv',
6504                        'msgsnd',
6505                        'munlock',
6506                        'munlockall',
6507                        'old_readdir',
6508                        'open',
6509                        'open_by_handle_at',
6510                        'openat',
6511                        'perf_event_open',
6512                        'pipe2',
6513                        'pivot_root',
6514                        'pkey_mprotect',
6515                        'prctl',
6516                        'pread64',
6517                        'preadv',
6518                        'preadv2',
6519                        'preadv64',
6520                        'preadv64v2',
6521                        'prlimit64',
6522                        'ptrace',
6523                        'pwrite64',
6524                        'pwritev',
6525                        'pwritev2',
6526                        'pwritev64',
6527                        'pwritev64v2',
```



```
6528                        'quotactl',
6529                        'read',
6530                        'readahead',
6531                        'readlinkat',
6532                        'readv',
6533                        'reboot',
6534                        'recvfrom',
6535                        'remap_file_pages',
6536                        'renameat2',
6537                        'request_key',
6538                        'rmdir',
6539                        'rt_sigaction',
6540                        'rt_sigprocmask',
6541                        'rt_sigtimedwait',
6542                        'sched_getaffinity',
6543                        'sched_getattr',
6544                        'sched_getparam',
6545                        'sched_getscheduler',
6546                        'sched_rr_get_interval',
6547                        'sched_setaffinity',
6548                        'sched_setattr',
6549                        'sched_setparam',
6550                        'sched_setscheduler',
6551                        'sched_yield',
6552                        'semctl',
6553                        'semtimedop',
6554                        'sendfile',
6555                        'sendfile64',
6556                        'sendto',
6557                        'set_trip_temp',
6558                        'setgid',
6559                        'setgroups',
6560                        'setgroups16',
6561                        'setitimer',
6562                        'setns',
6563                        'setpgid',
6564                        'setpriority',
6565                        'setregid',
6566                        'setresgid',
6567                        'setresuid',
6568                        'setreuid',
6569                        'setsid',
6570                        'setsockopt',
6571                        'setuid',
6572                        'shmat',
6573                        'shmctl',
6574                        'shmdt',
6575                        'shutdown',
6576                        'sigaction',
6577                        'sigaltstack',
6578                        'signal',
6579                        'signalfd4',
6580                        'socket',
6581                        'socketpair',
6582                        'splice',
6583                        'swapoff',
6584                        'swapon',
6585                        'symlinkat',
6586                        'sync_file_range',
6587                        'syncfs',
6588                        'tee',
6589                        'timer_create',
6590                        'timer_delete',
6591                        'timer_getoverrun',
6592                        'timer_gettime',
6593                        'timer_settime',
```

```
                        'timerfd_create',
                        'timerfd_gettime',
                        'timerfd_settime',
                        'umount',
                        'unlink',
                        'unlinkat',
                        'unshare',
                        'uselib',
                        'ustat',
                        'utime',
                        'utimensat',
                        'vfork',
                        'vmsplice',
                        'write',
                        'writev']),
 'pivot_root': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'ftruncate',
                    'getcwd',
                    'linkat',
                    'lookup_dcookie',
                    'memfd_create',
                    'mkdirat',
                    'mknodat',
                    'mmap_pgoff',
                    'mq_open',
                    'mq_unlink',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pivot_root',
                    'quotactl',
                    'remap_file_pages',
                    'renameat2',
                    'rmdir',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'symlinkat',
                    'umount',
                    'unlink',
                    'unlinkat',
                    'unshare',
                    'uselib']),
 'pkey_alloc': set(['brk',
                    'get_mempolicy',
                    'getrusage',
                    'io_cancel',
                    'io_destroy',
                    'io_getevents',
                    'io_setup',
                    'mbind',
                    'migrate_pages',
```



```
                    'mincore',
                    'modify_ldt',
                    'move_pages',
                    'mremap',
                    'pkey_alloc',
                    'pkey_free',
                    'prctl',
                    'remap_file_pages',
                    'shmdt',
                    'swapoff']),
 'pkey_mprotect': set(['brk',
                       'capget',
                       'clone',
                       'fork',
                       'get_mempolicy',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'madvise',
                       'migrate_pages',
                       'mincore',
                       'mlockall',
                       'move_pages',
                       'mprotect',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'mremap',
                       'msgrcv',
                       'munlock',
                       'munlockall',
                       'perf_event_open',
                       'personality',
                       'pkey_mprotect',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'remap_file_pages',
                       'rt_sigaction',
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getattr',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setattr',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'shmdt',
                       'sigaction',
                       'sigaltstack',
```

```
                    'signal',
                    'umount',
                    'vfork']),
 'poll': set(['poll', 'ppoll']),
 'ppoll': set(['capget',
               'clock_nanosleep',
               'clone',
               'epoll_wait',
               'faccessat',
               'fchmod',
               'fchmodat',
               'fchown',
               'fchownat',
               'fork',
               'fstat',
               'ftruncate',
               'futex',
               'futimesat',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'inotify_add_watch',
               'io_getevents',
               'ioctl',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'linkat',
               'memfd_create',
               'migrate_pages',
               'move_pages',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'mq_unlink',
               'msgrcv',
               'nanosleep',
               'newfstat',
               'perf_event_open',
               'personality',
               'poll',
               'ppoll',
               'prctl',
               'prlimit64',
               'pselect6',
               'ptrace',
               'readlinkat',
               'recvmmsg',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
```



```
               'select',
               'semtimedop',
               'sendfile',
               'sendfile64',
               'setitimer',
               'setns',
               'setpgid',
               'setpriority',
               'setsid',
               'settimeofday',
               'sigaction',
               'sigaltstack',
               'signal',
               'stime',
               'swapoff',
               'swapon',
               'timer_gettime',
               'timer_settime',
               'timerfd_gettime',
               'timerfd_settime',
               'umount',
               'unlink',
               'unlinkat',
               'uselib',
               'utime',
               'vfork']),
 'prctl': set(['brk',
               'capget',
               'clone',
               'fork',
               'get_mempolicy',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'io_cancel',
               'io_destroy',
               'io_getevents',
               'io_setup',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'mbind',
               'migrate_pages',
               'mincore',
               'modify_ldt',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'mremap',
               'msgrcv',
               'perf_event_open',
               'personality',
               'prctl',
               'prlimit64',
               'ptrace',
               'remap_file_pages',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
```

```
6858                 'sched_getparam',
6859                 'sched_getscheduler',
6860                 'sched_rr_get_interval',
6861                 'sched_setaffinity',
6862                 'sched_setattr',
6863                 'sched_setparam',
6864                 'sched_setscheduler',
6865                 'semtimedop',
6866                 'setitimer',
6867                 'setns',
6868                 'setpgid',
6869                 'setpriority',
6870                 'setresuid',
6871                 'setreuid',
6872                 'setsid',
6873                 'setuid',
6874                 'shmdt',
6875                 'sigaction',
6876                 'sigaltstack',
6877                 'signal',
6878                 'swapoff',
6879                 'umount',
6880                 'vfork']),
6881  'pread64': set(['accept4',
6882                  'acct',
6883                  'bind',
6884                  'bpf',
6885                  'connect',
6886                  'copy_file_range',
6887                  'dup',
6888                  'dup3',
6889                  'epoll_create1',
6890                  'epoll_ctl',
6891                  'epoll_wait',
6892                  'eventfd2',
6893                  'fallocate',
6894                  'fchdir',
6895                  'fchmod',
6896                  'fchown',
6897                  'fcntl',
6898                  'fcntl64',
6899                  'fdatasync',
6900                  'fgetxattr',
6901                  'flistxattr',
6902                  'flock',
6903                  'fremovexattr',
6904                  'fsetxattr',
6905                  'fstatfs',
6906                  'fstatfs64',
6907                  'fsync',
6908                  'ftruncate',
6909                  'futimesat',
6910                  'getdents',
6911                  'getdents64',
6912                  'getpeername',
6913                  'getsockname',
6914                  'getsockopt',
6915                  'inotify_add_watch',
6916                  'inotify_rm_watch',
6917                  'ioctl',
6918                  'listen',
6919                  'llseek',
6920                  'lseek',
6921                  'memfd_create',
6922                  'mmap_pgoff',
6923                  'mq_getsetattr',
```



```
6924                  'mq_notify',
6925                  'mq_open',
6926                  'mq_timedreceive',
6927                  'mq_timedsend',
6928                  'old_readdir',
6929                  'open',
6930                  'open_by_handle_at',
6931                  'openat',
6932                  'perf_event_open',
6933                  'pipe2',
6934                  'pread64',
6935                  'preadv',
6936                  'preadv2',
6937                  'preadv64',
6938                  'preadv64v2',
6939                  'pwrite64',
6940                  'pwritev',
6941                  'pwritev2',
6942                  'pwritev64',
6943                  'pwritev64v2',
6944                  'read',
6945                  'readahead',
6946                  'readv',
6947                  'recvfrom',
6948                  'remap_file_pages',
6949                  'sendfile',
6950                  'sendfile64',
6951                  'sendto',
6952                  'setns',
6953                  'setsockopt',
6954                  'shmat',
6955                  'shmctl',
6956                  'shmdt',
6957                  'shutdown',
6958                  'signalfd4',
6959                  'socket',
6960                  'socketpair',
6961                  'splice',
6962                  'swapoff',
6963                  'swapon',
6964                  'sync_file_range',
6965                  'syncfs',
6966                  'tee',
6967                  'uselib',
6968                  'utime',
6969                  'utimensat',
6970                  'vmsplice',
6971                  'write',
6972                  'writev']),
6973  'preadv': set(['accept4',
6974                 'acct',
6975                 'bind',
6976                 'bpf',
6977                 'connect',
6978                 'copy_file_range',
6979                 'dup',
6980                 'dup3',
6981                 'epoll_create1',
6982                 'epoll_ctl',
6983                 'epoll_wait',
6984                 'eventfd2',
6985                 'fallocate',
6986                 'fchdir',
6987                 'fchmod',
6988                 'fchown',
6989                 'fcntl',
```

```
6990                 'fcntl64',
6991                 'fdatasync',
6992                 'fgetxattr',
6993                 'flistxattr',
6994                 'flock',
6995                 'fremovexattr',
6996                 'fsetxattr',
6997                 'fstatfs',
6998                 'fstatfs64',
6999                 'fsync',
7000                 'ftruncate',
7001                 'futimesat',
7002                 'getdents',
7003                 'getdents64',
7004                 'getpeername',
7005                 'getsockname',
7006                 'getsockopt',
7007                 'inotify_add_watch',
7008                 'inotify_rm_watch',
7009                 'ioctl',
7010                 'listen',
7011                 'llseek',
7012                 'lseek',
7013                 'memfd_create',
7014                 'mmap_pgoff',
7015                 'mq_getsetattr',
7016                 'mq_notify',
7017                 'mq_open',
7018                 'mq_timedreceive',
7019                 'mq_timedsend',
7020                 'old_readdir',
7021                 'open',
7022                 'open_by_handle_at',
7023                 'openat',
7024                 'perf_event_open',
7025                 'pipe2',
7026                 'pread64',
7027                 'preadv',
7028                 'preadv2',
7029                 'preadv64',
7030                 'preadv64v2',
7031                 'pwrite64',
7032                 'pwritev',
7033                 'pwritev2',
7034                 'pwritev64',
7035                 'pwritev64v2',
7036                 'read',
7037                 'readahead',
7038                 'readv',
7039                 'recvfrom',
7040                 'remap_file_pages',
7041                 'sendfile',
7042                 'sendfile64',
7043                 'sendto',
7044                 'setns',
7045                 'setsockopt',
7046                 'shmat',
7047                 'shmctl',
7048                 'shmdt',
7049                 'shutdown',
7050                 'signalfd4',
7051                 'socket',
7052                 'socketpair',
7053                 'splice',
7054                 'swapoff',
7055                 'swapon',
```



```
7056                 'sync_file_range',
7057                 'syncfs',
7058                 'tee',
7059                 'uselib',
7060                 'utime',
7061                 'utimensat',
7062                 'vmsplice',
7063                 'write',
7064                 'writev']),
7065  'preadv2': set(['accept4',
7066                  'acct',
7067                  'bind',
7068                  'bpf',
7069                  'connect',
7070                  'copy_file_range',
7071                  'dup',
7072                  'dup3',
7073                  'epoll_create1',
7074                  'epoll_ctl',
7075                  'epoll_wait',
7076                  'eventfd2',
7077                  'fallocate',
7078                  'fchdir',
7079                  'fchmod',
7080                  'fchown',
7081                  'fcntl',
7082                  'fcntl64',
7083                  'fdatasync',
7084                  'fgetxattr',
7085                  'flistxattr',
7086                  'flock',
7087                  'fremovexattr',
7088                  'fsetxattr',
7089                  'fstatfs',
7090                  'fstatfs64',
7091                  'fsync',
7092                  'ftruncate',
7093                  'futimesat',
7094                  'getdents',
7095                  'getdents64',
7096                  'getpeername',
7097                  'getsockname',
7098                  'getsockopt',
7099                  'inotify_add_watch',
7100                  'inotify_rm_watch',
7101                  'ioctl',
7102                  'listen',
7103                  'llseek',
7104                  'lseek',
7105                  'memfd_create',
7106                  'mmap_pgoff',
7107                  'mq_getsetattr',
7108                  'mq_notify',
7109                  'mq_open',
7110                  'mq_timedreceive',
7111                  'mq_timedsend',
7112                  'old_readdir',
7113                  'open',
7114                  'open_by_handle_at',
7115                  'openat',
7116                  'perf_event_open',
7117                  'pipe2',
7118                  'pread64',
7119                  'preadv',
7120                  'preadv2',
7121                  'preadv64',
```

```
                  'preadv64v2',
                  'pwrite64',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'remap_file_pages',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setns',
                  'setsockopt',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'shutdown',
                  'signalfd4',
                  'socket',
                  'socketpair',
                  'splice',
                  'swapoff',
                  'swapon',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'uselib',
                  'utime',
                  'utimensat',
                  'vmsplice',
                  'write',
                  'writev']),
 'preadv64': set(['accept4',
                  'acct',
                  'bind',
                  'bpf',
                  'connect',
                  'copy_file_range',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'epoll_wait',
                  'eventfd2',
                  'fallocate',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
                  'fremovexattr',
                  'fsetxattr',
                  'fstatfs',
                  'fstatfs64',
                  'fsync',
                  'ftruncate',
                  'futimesat',
                  'getdents',
                  'getdents64',
```



```
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'inotify_rm_watch',
                  'ioctl',
                  'listen',
                  'llseek',
                  'lseek',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_open',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'old_readdir',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pread64',
                  'preadv',
                  'preadv2',
                  'preadv64',
                  'preadv64v2',
                  'pwrite64',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'remap_file_pages',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setns',
                  'setsockopt',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'shutdown',
                  'signalfd4',
                  'socket',
                  'socketpair',
                  'splice',
                  'swapoff',
                  'swapon',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'uselib',
                  'utime',
                  'utimensat',
                  'vmsplice',
                  'write',
                  'writev']),
 'preadv64v2': set(['accept4',
                    'acct',
                    'bind',
                    'bpf',
                    'connect',
```

```
                    'copy_file_range',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'epoll_wait',
                    'eventfd2',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'sendto',
```



```
                    'setns',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'prlimit64': set(['capget',
                   'capset',
                   'clone',
                   'epoll_create1',
                   'faccessat',
                   'fork',
                   'get_robust_list',
                   'getgroups',
                   'getgroups16',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getresgid',
                   'getresgid16',
                   'getresuid',
                   'getresuid16',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'old_getrlimit',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
```

```
                    'semtimedop',
                    'setfsgid',
                    'setfsuid',
                    'setgid',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setregid',
                    'setresgid',
                    'setresuid',
                    'setreuid',
                    'setrlimit',
                    'setsid',
                    'setuid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'unshare',
                    'vfork']),
 'pselect6': set(['capget',
                  'clock_nanosleep',
                  'clone',
                  'epoll_wait',
                  'faccessat',
                  'fchmod',
                  'fchmodat',
                  'fchown',
                  'fchownat',
                  'fork',
                  'fstat',
                  'ftruncate',
                  'futex',
                  'futimesat',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'inotify_add_watch',
                  'io_getevents',
                  'ioctl',
                  'ioperm',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'linkat',
                  'memfd_create',
                  'migrate_pages',
                  'move_pages',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mq_unlink',
                  'msgrcv',
                  'nanosleep',
                  'newfstat',
                  'perf_event_open',
                  'poll',
                  'ppoll',
                  'prctl',
```



```
                    'prlimit64',
                    'pselect6',
                    'ptrace',
                    'readlinkat',
                    'recvmmsg',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'select',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'settimeofday',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'stime',
                    'swapoff',
                    'swapon',
                    'timer_gettime',
                    'timer_settime',
                    'timerfd_gettime',
                    'timerfd_settime',
                    'umount',
                    'unlink',
                    'unlinkat',
                    'uselib',
                    'utime',
                    'vfork']),
 'ptrace': set(['capget',
                'clone',
                'epoll_wait',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'pause',
                'perf_event_open',
                'prctl',
```

```
7518                 'prlimit64',
7519                 'ptrace',
7520                 'rt_sigaction',
7521                 'rt_sigprocmask',
7522                 'rt_sigsuspend',
7523                 'rt_sigtimedwait',
7524                 'sched_getaffinity',
7525                 'sched_getattr',
7526                 'sched_getparam',
7527                 'sched_getscheduler',
7528                 'sched_rr_get_interval',
7529                 'sched_setaffinity',
7530                 'sched_setattr',
7531                 'sched_setparam',
7532                 'sched_setscheduler',
7533                 'semtimedop',
7534                 'setitimer',
7535                 'setns',
7536                 'setpgid',
7537                 'setpriority',
7538                 'setresuid',
7539                 'setreuid',
7540                 'setsid',
7541                 'setuid',
7542                 'sigaction',
7543                 'sigaltstack',
7544                 'signal',
7545                 'sigsuspend',
7546                 'umount',
7547                 'vfork']),
7548  'pwrite64': set(['accept4',
7549                   'acct',
7550                   'bind',
7551                   'bpf',
7552                   'connect',
7553                   'copy_file_range',
7554                   'dup',
7555                   'dup3',
7556                   'epoll_create1',
7557                   'epoll_ctl',
7558                   'epoll_wait',
7559                   'eventfd2',
7560                   'fallocate',
7561                   'fchdir',
7562                   'fchmod',
7563                   'fchown',
7564                   'fcntl',
7565                   'fcntl64',
7566                   'fdatasync',
7567                   'fgetxattr',
7568                   'flistxattr',
7569                   'flock',
7570                   'fremovexattr',
7571                   'fsetxattr',
7572                   'fstatfs',
7573                   'fstatfs64',
7574                   'fsync',
7575                   'ftruncate',
7576                   'futimesat',
7577                   'getdents',
7578                   'getdents64',
7579                   'getpeername',
7580                   'getsockname',
7581                   'getsockopt',
7582                   'inotify_add_watch',
7583                   'inotify_rm_watch',
```



```
7584                 'ioctl',
7585                 'listen',
7586                 'llseek',
7587                 'lseek',
7588                 'memfd_create',
7589                 'mmap_pgoff',
7590                 'mq_getsetattr',
7591                 'mq_notify',
7592                 'mq_open',
7593                 'mq_timedreceive',
7594                 'mq_timedsend',
7595                 'old_readdir',
7596                 'open',
7597                 'open_by_handle_at',
7598                 'openat',
7599                 'perf_event_open',
7600                 'pipe2',
7601                 'pread64',
7602                 'preadv',
7603                 'preadv2',
7604                 'preadv64',
7605                 'preadv64v2',
7606                 'pwrite64',
7607                 'pwritev',
7608                 'pwritev2',
7609                 'pwritev64',
7610                 'pwritev64v2',
7611                 'read',
7612                 'readahead',
7613                 'readv',
7614                 'recvfrom',
7615                 'remap_file_pages',
7616                 'sendfile',
7617                 'sendfile64',
7618                 'sendto',
7619                 'setns',
7620                 'setsockopt',
7621                 'shmat',
7622                 'shmctl',
7623                 'shmdt',
7624                 'shutdown',
7625                 'signalfd4',
7626                 'socket',
7627                 'socketpair',
7628                 'splice',
7629                 'swapoff',
7630                 'swapon',
7631                 'sync_file_range',
7632                 'syncfs',
7633                 'tee',
7634                 'uselib',
7635                 'utime',
7636                 'utimensat',
7637                 'vmsplice',
7638                 'write',
7639                 'writev']),
7640  'pwritev': set(['accept4',
7641                  'acct',
7642                  'bind',
7643                  'bpf',
7644                  'connect',
7645                  'copy_file_range',
7646                  'dup',
7647                  'dup3',
7648                  'epoll_create1',
7649                  'epoll_ctl',
```

```
7650                 'epoll_wait',
7651                 'eventfd2',
7652                 'fallocate',
7653                 'fchdir',
7654                 'fchmod',
7655                 'fchown',
7656                 'fcntl',
7657                 'fcntl64',
7658                 'fdatasync',
7659                 'fgetxattr',
7660                 'flistxattr',
7661                 'flock',
7662                 'fremovexattr',
7663                 'fsetxattr',
7664                 'fstatfs',
7665                 'fstatfs64',
7666                 'fsync',
7667                 'ftruncate',
7668                 'futimesat',
7669                 'getdents',
7670                 'getdents64',
7671                 'getpeername',
7672                 'getsockname',
7673                 'getsockopt',
7674                 'inotify_add_watch',
7675                 'inotify_rm_watch',
7676                 'ioctl',
7677                 'listen',
7678                 'llseek',
7679                 'lseek',
7680                 'memfd_create',
7681                 'mmap_pgoff',
7682                 'mq_getsetattr',
7683                 'mq_notify',
7684                 'mq_open',
7685                 'mq_timedreceive',
7686                 'mq_timedsend',
7687                 'old_readdir',
7688                 'open',
7689                 'open_by_handle_at',
7690                 'openat',
7691                 'perf_event_open',
7692                 'pipe2',
7693                 'pread64',
7694                 'preadv',
7695                 'preadv2',
7696                 'preadv64',
7697                 'preadv64v2',
7698                 'pwrite64',
7699                 'pwritev',
7700                 'pwritev2',
7701                 'pwritev64',
7702                 'pwritev64v2',
7703                 'read',
7704                 'readahead',
7705                 'readv',
7706                 'recvfrom',
7707                 'remap_file_pages',
7708                 'sendfile',
7709                 'sendfile64',
7710                 'sendto',
7711                 'setns',
7712                 'setsockopt',
7713                 'shmat',
7714                 'shmctl',
7715                 'shmdt',
```



```
7716                 'shutdown',
7717                 'signalfd4',
7718                 'socket',
7719                 'socketpair',
7720                 'splice',
7721                 'swapoff',
7722                 'swapon',
7723                 'sync_file_range',
7724                 'syncfs',
7725                 'tee',
7726                 'uselib',
7727                 'utime',
7728                 'utimensat',
7729                 'vmsplice',
7730                 'write',
7731                 'writev']),
7732  'pwritev2': set(['accept4',
7733                   'acct',
7734                   'bind',
7735                   'bpf',
7736                   'connect',
7737                   'copy_file_range',
7738                   'dup',
7739                   'dup3',
7740                   'epoll_create1',
7741                   'epoll_ctl',
7742                   'epoll_wait',
7743                   'eventfd2',
7744                   'fallocate',
7745                   'fchdir',
7746                   'fchmod',
7747                   'fchown',
7748                   'fcntl',
7749                   'fcntl64',
7750                   'fdatasync',
7751                   'fgetxattr',
7752                   'flistxattr',
7753                   'flock',
7754                   'fremovexattr',
7755                   'fsetxattr',
7756                   'fstatfs',
7757                   'fstatfs64',
7758                   'fsync',
7759                   'ftruncate',
7760                   'futimesat',
7761                   'getdents',
7762                   'getdents64',
7763                   'getpeername',
7764                   'getsockname',
7765                   'getsockopt',
7766                   'inotify_add_watch',
7767                   'inotify_rm_watch',
7768                   'ioctl',
7769                   'listen',
7770                   'llseek',
7771                   'lseek',
7772                   'memfd_create',
7773                   'mmap_pgoff',
7774                   'mq_getsetattr',
7775                   'mq_notify',
7776                   'mq_open',
7777                   'mq_timedreceive',
7778                   'mq_timedsend',
7779                   'old_readdir',
7780                   'open',
7781                   'open_by_handle_at',
```

```
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pread64',
                  'preadv',
                  'preadv2',
                  'preadv64',
                  'preadv64v2',
                  'pwrite64',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'remap_file_pages',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setns',
                  'setsockopt',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'shutdown',
                  'signalfd4',
                  'socket',
                  'socketpair',
                  'splice',
                  'swapoff',
                  'swapon',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'uselib',
                  'utime',
                  'utimensat',
                  'vmsplice',
                  'write',
                  'writev']),
 'pwritev64': set(['accept4',
                   'acct',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'epoll_wait',
                   'eventfd2',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'fgetxattr',
                   'flistxattr',
                   'flock',
                   'fremovexattr',
                   'fsetxattr',
```



```
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'getdents',
                   'getdents64',
                   'getpeername',
                   'getsockname',
                   'getsockopt',
                   'inotify_add_watch',
                   'inotify_rm_watch',
                   'ioctl',
                   'listen',
                   'llseek',
                   'lseek',
                   'memfd_create',
                   'mmap_pgoff',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_open',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'old_readdir',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'pread64',
                   'preadv',
                   'preadv2',
                   'preadv64',
                   'preadv64v2',
                   'pwrite64',
                   'pwritev',
                   'pwritev2',
                   'pwritev64',
                   'pwritev64v2',
                   'read',
                   'readahead',
                   'readv',
                   'recvfrom',
                   'remap_file_pages',
                   'sendfile',
                   'sendfile64',
                   'sendto',
                   'setns',
                   'setsockopt',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'shutdown',
                   'signalfd4',
                   'socket',
                   'socketpair',
                   'splice',
                   'swapoff',
                   'swapon',
                   'sync_file_range',
                   'syncfs',
                   'tee',
                   'uselib',
                   'utime',
                   'utimensat',
                   'vmsplice',
```

```
7914                    'write',
7915                    'writev']),
7916  'pwritev64v2': set(['accept4',
7917                      'acct',
7918                      'bind',
7919                      'bpf',
7920                      'connect',
7921                      'copy_file_range',
7922                      'dup',
7923                      'dup3',
7924                      'epoll_create1',
7925                      'epoll_ctl',
7926                      'epoll_wait',
7927                      'eventfd2',
7928                      'fallocate',
7929                      'fchdir',
7930                      'fchmod',
7931                      'fchown',
7932                      'fcntl',
7933                      'fcntl64',
7934                      'fdatasync',
7935                      'fgetxattr',
7936                      'flistxattr',
7937                      'flock',
7938                      'fremovexattr',
7939                      'fsetxattr',
7940                      'fstatfs',
7941                      'fstatfs64',
7942                      'fsync',
7943                      'ftruncate',
7944                      'futimesat',
7945                      'getdents',
7946                      'getdents64',
7947                      'getpeername',
7948                      'getsockname',
7949                      'getsockopt',
7950                      'inotify_add_watch',
7951                      'inotify_rm_watch',
7952                      'ioctl',
7953                      'listen',
7954                      'llseek',
7955                      'lseek',
7956                      'memfd_create',
7957                      'mmap_pgoff',
7958                      'mq_getsetattr',
7959                      'mq_notify',
7960                      'mq_open',
7961                      'mq_timedreceive',
7962                      'mq_timedsend',
7963                      'old_readdir',
7964                      'open',
7965                      'open_by_handle_at',
7966                      'openat',
7967                      'perf_event_open',
7968                      'pipe2',
7969                      'pread64',
7970                      'preadv',
7971                      'preadv2',
7972                      'preadv64',
7973                      'preadv64v2',
7974                      'pwrite64',
7975                      'pwritev',
7976                      'pwritev2',
7977                      'pwritev64',
7978                      'pwritev64v2',
7979                      'read',
```



```
7980                      'readahead',
7981                      'readv',
7982                      'recvfrom',
7983                      'remap_file_pages',
7984                      'sendfile',
7985                      'sendfile64',
7986                      'sendto',
7987                      'setns',
7988                      'setsockopt',
7989                      'shmat',
7990                      'shmctl',
7991                      'shmdt',
7992                      'shutdown',
7993                      'signalfd4',
7994                      'socket',
7995                      'socketpair',
7996                      'splice',
7997                      'swapoff',
7998                      'swapon',
7999                      'sync_file_range',
8000                      'syncfs',
8001                      'tee',
8002                      'uselib',
8003                      'utime',
8004                      'utimensat',
8005                      'vmsplice',
8006                      'write',
8007                      'writev']),
8008  'quotactl': set(['acct',
8009                   'mq_open',
8010                   'mq_unlink',
8011                   'open',
8012                   'openat',
8013                   'quotactl',
8014                   'renameat2',
8015                   'rmdir',
8016                   'swapoff',
8017                   'swapon',
8018                   'symlinkat',
8019                   'syncfs',
8020                   'sysfs',
8021                   'umount',
8022                   'unlink',
8023                   'unlinkat',
8024                   'uselib',
8025                   'ustat']),
8026  'read': set(['accept4',
8027               'bind',
8028               'bpf',
8029               'connect',
8030               'copy_file_range',
8031               'epoll_ctl',
8032               'epoll_wait',
8033               'fallocate',
8034               'fchdir',
8035               'fchmod',
8036               'fchown',
8037               'fcntl',
8038               'fcntl64',
8039               'fdatasync',
8040               'fgetxattr',
8041               'flistxattr',
8042               'flock',
8043               'fremovexattr',
8044               'fsetxattr',
8045               'fstatfs',
```

```
8046                 'fstatfs64',
8047                 'fsync',
8048                 'ftruncate',
8049                 'futimesat',
8050                 'getdents',
8051                 'getdents64',
8052                 'getpeername',
8053                 'getsockname',
8054                 'getsockopt',
8055                 'inotify_add_watch',
8056                 'inotify_rm_watch',
8057                 'ioctl',
8058                 'listen',
8059                 'llseek',
8060                 'lseek',
8061                 'mq_getsetattr',
8062                 'mq_notify',
8063                 'mq_timedreceive',
8064                 'mq_timedsend',
8065                 'old_readdir',
8066                 'perf_event_open',
8067                 'pread64',
8068                 'preadv',
8069                 'preadv2',
8070                 'preadv64',
8071                 'preadv64v2',
8072                 'pwrite64',
8073                 'pwritev',
8074                 'pwritev2',
8075                 'pwritev64',
8076                 'pwritev64v2',
8077                 'read',
8078                 'readahead',
8079                 'readv',
8080                 'recvfrom',
8081                 'sendfile',
8082                 'sendfile64',
8083                 'sendto',
8084                 'setsockopt',
8085                 'shutdown',
8086                 'signalfd4',
8087                 'splice',
8088                 'sync_file_range',
8089                 'syncfs',
8090                 'tee',
8091                 'utime',
8092                 'utimensat',
8093                 'vmsplice',
8094                 'write',
8095                 'writev']),
8096  'readahead': set(['accept4',
8097                    'acct',
8098                    'bind',
8099                    'bpf',
8100                    'connect',
8101                    'copy_file_range',
8102                    'dup',
8103                    'dup3',
8104                    'epoll_create1',
8105                    'epoll_ctl',
8106                    'epoll_wait',
8107                    'eventfd2',
8108                    'faccessat',
8109                    'fallocate',
8110                    'fchdir',
8111                    'fchmod',
```



```
8112                    'fchmodat',
8113                    'fchown',
8114                    'fchownat',
8115                    'fcntl',
8116                    'fcntl64',
8117                    'fdatasync',
8118                    'fgetxattr',
8119                    'flistxattr',
8120                    'flock',
8121                    'fremovexattr',
8122                    'fsetxattr',
8123                    'fstatfs',
8124                    'fstatfs64',
8125                    'fsync',
8126                    'ftruncate',
8127                    'futimesat',
8128                    'getdents',
8129                    'getdents64',
8130                    'getpeername',
8131                    'getsockname',
8132                    'getsockopt',
8133                    'inotify_add_watch',
8134                    'inotify_rm_watch',
8135                    'ioctl',
8136                    'linkat',
8137                    'listen',
8138                    'llseek',
8139                    'lseek',
8140                    'memfd_create',
8141                    'mmap_pgoff',
8142                    'mq_getsetattr',
8143                    'mq_notify',
8144                    'mq_open',
8145                    'mq_timedreceive',
8146                    'mq_timedsend',
8147                    'mq_unlink',
8148                    'old_readdir',
8149                    'open',
8150                    'open_by_handle_at',
8151                    'openat',
8152                    'perf_event_open',
8153                    'pipe2',
8154                    'pread64',
8155                    'preadv',
8156                    'preadv2',
8157                    'preadv64',
8158                    'preadv64v2',
8159                    'pwrite64',
8160                    'pwritev',
8161                    'pwritev2',
8162                    'pwritev64',
8163                    'pwritev64v2',
8164                    'read',
8165                    'readahead',
8166                    'readlinkat',
8167                    'readv',
8168                    'recvfrom',
8169                    'remap_file_pages',
8170                    'sendfile',
8171                    'sendfile64',
8172                    'sendto',
8173                    'setns',
8174                    'setsockopt',
8175                    'shmat',
8176                    'shmctl',
8177                    'shmdt',
```

```
                   'shutdown',
                   'signalfd4',
                   'socket',
                   'socketpair',
                   'splice',
                   'swapoff',
                   'swapon',
                   'sync_file_range',
                   'syncfs',
                   'tee',
                   'unlink',
                   'unlinkat',
                   'uselib',
                   'utime',
                   'utimensat',
                   'vmsplice',
                   'write',
                   'writev']),
 'readlinkat': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'getcwd',
                    'lookup_dcookie',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_open',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pivot_root',
                    'quotactl',
                    'remap_file_pages',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'unshare',
                    'uselib']),
 'readv': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
```



```
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'sendfile',
               'sendfile64',
               'sendto',
               'setsockopt',
               'shutdown',
               'signalfd4',
               'splice',
               'sync_file_range',
               'syncfs',
               'tee',
               'utime',
               'utimensat',
               'vmsplice',
               'write',
               'writev']),
 'reboot': set(['acct', 'perf_event_open', 'reboot', 'setns']),
 'recvfrom': set(['accept4',
                  'acct',
                  'bind',
                  'bpf',
                  'connect',
                  'copy_file_range',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'epoll_wait',
                  'eventfd2',
```

```
8310                    'fallocate',
8311                    'fchdir',
8312                    'fchmod',
8313                    'fchown',
8314                    'fcntl',
8315                    'fcntl64',
8316                    'fdatasync',
8317                    'fgetxattr',
8318                    'flistxattr',
8319                    'flock',
8320                    'fremovexattr',
8321                    'fsetxattr',
8322                    'fstatfs',
8323                    'fstatfs64',
8324                    'fsync',
8325                    'ftruncate',
8326                    'futimesat',
8327                    'getdents',
8328                    'getdents64',
8329                    'getpeername',
8330                    'getsockname',
8331                    'getsockopt',
8332                    'inotify_add_watch',
8333                    'inotify_rm_watch',
8334                    'ioctl',
8335                    'listen',
8336                    'llseek',
8337                    'lseek',
8338                    'memfd_create',
8339                    'mmap_pgoff',
8340                    'mq_getsetattr',
8341                    'mq_notify',
8342                    'mq_open',
8343                    'mq_timedreceive',
8344                    'mq_timedsend',
8345                    'old_readdir',
8346                    'open',
8347                    'open_by_handle_at',
8348                    'openat',
8349                    'perf_event_open',
8350                    'pipe2',
8351                    'pread64',
8352                    'preadv',
8353                    'preadv2',
8354                    'preadv64',
8355                    'preadv64v2',
8356                    'pwrite64',
8357                    'pwritev',
8358                    'pwritev2',
8359                    'pwritev64',
8360                    'pwritev64v2',
8361                    'read',
8362                    'readahead',
8363                    'readv',
8364                    'recvfrom',
8365                    'remap_file_pages',
8366                    'sendfile',
8367                    'sendfile64',
8368                    'sendto',
8369                    'setns',
8370                    'setsockopt',
8371                    'shmat',
8372                    'shmctl',
8373                    'shmdt',
8374                    'shutdown',
8375                    'signalfd4',
```



```
8376                    'socket',
8377                    'socketpair',
8378                    'splice',
8379                    'swapoff',
8380                    'swapon',
8381                    'sync_file_range',
8382                    'syncfs',
8383                    'tee',
8384                    'uselib',
8385                    'utime',
8386                    'utimensat',
8387                    'vmsplice',
8388                    'write',
8389                    'writev']),
8390  'recvmmsg': set(['accept4',
8391                   'bind',
8392                   'clock_nanosleep',
8393                   'connect',
8394                   'epoll_wait',
8395                   'faccessat',
8396                   'fchmod',
8397                   'fchmodat',
8398                   'fchown',
8399                   'fchownat',
8400                   'fstat',
8401                   'ftruncate',
8402                   'futex',
8403                   'futimesat',
8404                   'getpeername',
8405                   'getsockname',
8406                   'getsockopt',
8407                   'inotify_add_watch',
8408                   'io_getevents',
8409                   'ioctl',
8410                   'linkat',
8411                   'listen',
8412                   'memfd_create',
8413                   'mq_getsetattr',
8414                   'mq_notify',
8415                   'mq_timedreceive',
8416                   'mq_timedsend',
8417                   'mq_unlink',
8418                   'nanosleep',
8419                   'newfstat',
8420                   'poll',
8421                   'ppoll',
8422                   'pselect6',
8423                   'readlinkat',
8424                   'recvfrom',
8425                   'recvmmsg',
8426                   'recvmsg',
8427                   'rt_sigtimedwait',
8428                   'sched_rr_get_interval',
8429                   'select',
8430                   'semtimedop',
8431                   'sendfile',
8432                   'sendfile64',
8433                   'sendmmsg',
8434                   'sendmsg',
8435                   'sendto',
8436                   'setsockopt',
8437                   'settimeofday',
8438                   'shutdown',
8439                   'stime',
8440                   'swapoff',
8441                   'swapon',
```

```
                  'timer_gettime',
                  'timer_settime',
                  'timerfd_gettime',
                  'timerfd_settime',
                  'unlink',
                  'unlinkat',
                  'uselib',
                  'utime']),
 'recvmsg': set(['accept4',
                 'bind',
                 'connect',
                 'getpeername',
                 'getsockname',
                 'getsockopt',
                 'listen',
                 'recvfrom',
                 'recvmmsg',
                 'recvmsg',
                 'sendmmsg',
                 'sendmsg',
                 'sendto',
                 'setsockopt',
                 'shutdown']),
 'remap_file_pages': set(['brk',
                          'get_mempolicy',
                          'madvise',
                          'mincore',
                          'mlockall',
                          'mprotect',
                          'mremap',
                          'munlock',
                          'munlockall',
                          'pkey_mprotect',
                          'prctl',
                          'remap_file_pages',
                          'shmdt']),
 'removexattr': set(['accept4',
                     'acct',
                     'dup',
                     'dup3',
                     'epoll_create1',
                     'epoll_ctl',
                     'eventfd2',
                     'flock',
                     'getcwd',
                     'lookup_dcookie',
                     'memfd_create',
                     'mmap_pgoff',
                     'mq_open',
                     'open',
                     'open_by_handle_at',
                     'openat',
                     'perf_event_open',
                     'pipe2',
                     'pivot_root',
                     'quotactl',
                     'remap_file_pages',
                     'setns',
                     'shmat',
                     'shmctl',
                     'shmdt',
                     'socket',
                     'socketpair',
                     'swapoff',
                     'swapon',
                     'unshare',
```



```
                     'uselib']),
 'renameat2': set(['accept4',
                   'acct',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'flock',
                   'ftruncate',
                   'getcwd',
                   'linkat',
                   'lookup_dcookie',
                   'memfd_create',
                   'mkdirat',
                   'mknodat',
                   'mmap_pgoff',
                   'mq_open',
                   'mq_unlink',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'pivot_root',
                   'quotactl',
                   'remap_file_pages',
                   'renameat2',
                   'rmdir',
                   'setns',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'socket',
                   'socketpair',
                   'swapoff',
                   'swapon',
                   'symlinkat',
                   'sysfs',
                   'unlink',
                   'unlinkat',
                   'unshare',
                   'uselib']),
 'rmdir': set(['accept4',
               'acct',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'eventfd2',
               'flock',
               'ftruncate',
               'getcwd',
               'linkat',
               'lookup_dcookie',
               'memfd_create',
               'mkdirat',
               'mknodat',
               'mmap_pgoff',
               'mq_open',
               'mq_unlink',
               'open',
               'open_by_handle_at',
               'openat',
               'perf_event_open',
               'pipe2',
```

```
8574                'pivot_root',
8575                'quotactl',
8576                'remap_file_pages',
8577                'renameat2',
8578                'rmdir',
8579                'setns',
8580                'shmat',
8581                'shmctl',
8582                'shmdt',
8583                'socket',
8584                'socketpair',
8585                'swapoff',
8586                'swapon',
8587                'symlinkat',
8588                'unlink',
8589                'unlinkat',
8590                'unshare',
8591                'uselib']),
8592  'rt_sigqueueinfo': set(['kill',
8593                          'rt_sigqueueinfo',
8594                          'rt_sigreturn',
8595                          'rt_sigtimedwait',
8596                          'rt_tgsigqueueinfo',
8597                          'tgkill',
8598                          'timer_create',
8599                          'tkill']),
8600  'rt_sigreturn': set(['capget',
8601                       'clone',
8602                       'fork',
8603                       'get_robust_list',
8604                       'getitimer',
8605                       'getpgid',
8606                       'getppid',
8607                       'getpriority',
8608                       'getrusage',
8609                       'getsid',
8610                       'ioperm',
8611                       'ioprio_get',
8612                       'ioprio_set',
8613                       'keyctl',
8614                       'kill',
8615                       'migrate_pages',
8616                       'move_pages',
8617                       'mq_timedreceive',
8618                       'mq_timedsend',
8619                       'msgrcv',
8620                       'perf_event_open',
8621                       'prctl',
8622                       'prlimit64',
8623                       'ptrace',
8624                       'rt_sigaction',
8625                       'rt_sigprocmask',
8626                       'rt_sigreturn',
8627                       'rt_sigtimedwait',
8628                       'sched_getaffinity',
8629                       'sched_getattr',
8630                       'sched_getparam',
8631                       'sched_getscheduler',
8632                       'sched_rr_get_interval',
8633                       'sched_setaffinity',
8634                       'sched_setattr',
8635                       'sched_setparam',
8636                       'sched_setscheduler',
8637                       'semtimedop',
8638                       'setitimer',
8639                       'setns',
```



```
8640                       'setpgid',
8641                       'setpriority',
8642                       'setsid',
8643                       'sigaction',
8644                       'sigaltstack',
8645                       'signal',
8646                       'umount',
8647                       'vfork']),
8648  'rt_sigtimedwait': set(['capget',
8649                          'clone',
8650                          'fork',
8651                          'get_robust_list',
8652                          'getitimer',
8653                          'getpgid',
8654                          'getppid',
8655                          'getpriority',
8656                          'getrusage',
8657                          'getsid',
8658                          'ioperm',
8659                          'ioprio_get',
8660                          'ioprio_set',
8661                          'keyctl',
8662                          'kill',
8663                          'migrate_pages',
8664                          'move_pages',
8665                          'mq_timedreceive',
8666                          'mq_timedsend',
8667                          'msgrcv',
8668                          'perf_event_open',
8669                          'prctl',
8670                          'prlimit64',
8671                          'ptrace',
8672                          'rt_sigaction',
8673                          'rt_sigprocmask',
8674                          'rt_sigqueueinfo',
8675                          'rt_sigreturn',
8676                          'rt_sigtimedwait',
8677                          'rt_tgsigqueueinfo',
8678                          'sched_getaffinity',
8679                          'sched_getattr',
8680                          'sched_getparam',
8681                          'sched_getscheduler',
8682                          'sched_rr_get_interval',
8683                          'sched_setaffinity',
8684                          'sched_setattr',
8685                          'sched_setparam',
8686                          'sched_setscheduler',
8687                          'semtimedop',
8688                          'setitimer',
8689                          'setns',
8690                          'setpgid',
8691                          'setpriority',
8692                          'setsid',
8693                          'sigaction',
8694                          'sigaltstack',
8695                          'signal',
8696                          'tgkill',
8697                          'timer_create',
8698                          'tkill',
8699                          'umount',
8700                          'vfork']),
8701  'rt_tgsigqueueinfo': set(['kill',
8702                            'rt_sigqueueinfo',
8703                            'rt_sigreturn',
8704                            'rt_sigtimedwait',
8705                            'rt_tgsigqueueinfo',
```

```
                        'tgkill',
                        'timer_create',
                        'tkill']),
 'sched_getattr': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioperm',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'migrate_pages',
                       'move_pages',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'msgrcv',
                       'perf_event_open',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'rt_sigaction',
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getattr',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setattr',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'sigaction',
                       'sigaltstack',
                       'signal',
                       'umount',
                       'vfork']),
 'sched_getparam': set(['capget',
                        'clone',
                        'fork',
                        'get_robust_list',
                        'getitimer',
                        'getpgid',
                        'getppid',
                        'getpriority',
                        'getrusage',
                        'getsid',
                        'ioprio_get',
                        'ioprio_set',
                        'keyctl',
                        'kill',
                        'migrate_pages',
                        'move_pages',
```



```
                        'mq_timedreceive',
                        'mq_timedsend',
                        'msgrcv',
                        'perf_event_open',
                        'prctl',
                        'prlimit64',
                        'ptrace',
                        'rt_sigaction',
                        'rt_sigprocmask',
                        'rt_sigtimedwait',
                        'sched_getaffinity',
                        'sched_getattr',
                        'sched_getparam',
                        'sched_getscheduler',
                        'sched_rr_get_interval',
                        'sched_setaffinity',
                        'sched_setattr',
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setsid',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'umount',
                        'vfork']),
 'sched_getscheduler': set(['capget',
                            'clone',
                            'fork',
                            'get_robust_list',
                            'getitimer',
                            'getpgid',
                            'getppid',
                            'getpriority',
                            'getrusage',
                            'getsid',
                            'ioprio_get',
                            'ioprio_set',
                            'keyctl',
                            'kill',
                            'migrate_pages',
                            'move_pages',
                            'mq_timedreceive',
                            'mq_timedsend',
                            'msgrcv',
                            'perf_event_open',
                            'prctl',
                            'prlimit64',
                            'ptrace',
                            'rt_sigaction',
                            'rt_sigprocmask',
                            'rt_sigtimedwait',
                            'sched_getaffinity',
                            'sched_getattr',
                            'sched_getparam',
                            'sched_getscheduler',
                            'sched_rr_get_interval',
                            'sched_setaffinity',
                            'sched_setattr',
                            'sched_setparam',
                            'sched_setscheduler',
                            'semtimedop',
```

```
                            'setitimer',
                            'setns',
                            'setpgid',
                            'setpriority',
                            'setsid',
                            'sigaction',
                            'sigaltstack',
                            'signal',
                            'umount',
                            'vfork']),
 'sched_rr_get_interval': set(['sched_rr_get_interval']),
 'sched_setaffinity': set(['capget',
                           'capset',
                           'clone',
                           'epoll_create1',
                           'faccessat',
                           'fork',
                           'get_robust_list',
                           'getgroups',
                           'getgroups16',
                           'getitimer',
                           'getpgid',
                           'getppid',
                           'getpriority',
                           'getresgid',
                           'getresgid16',
                           'getresuid',
                           'getresuid16',
                           'getrusage',
                           'getsid',
                           'ioprio_get',
                           'ioprio_set',
                           'keyctl',
                           'kill',
                           'migrate_pages',
                           'move_pages',
                           'mq_timedreceive',
                           'mq_timedsend',
                           'msgrcv',
                           'perf_event_open',
                           'prctl',
                           'prlimit64',
                           'ptrace',
                           'rt_sigaction',
                           'rt_sigprocmask',
                           'rt_sigtimedwait',
                           'sched_getaffinity',
                           'sched_getattr',
                           'sched_getparam',
                           'sched_getscheduler',
                           'sched_rr_get_interval',
                           'sched_setaffinity',
                           'sched_setattr',
                           'sched_setparam',
                           'sched_setscheduler',
                           'semtimedop',
                           'setfsgid',
                           'setfsuid',
                           'setgid',
                           'setitimer',
                           'setns',
                           'setpgid',
                           'setpriority',
                           'setregid',
                           'setresgid',
                           'setresuid',
```



```
                           'setreuid',
                           'setsid',
                           'setuid',
                           'sigaction',
                           'sigaltstack',
                           'signal',
                           'umount',
                           'unshare',
                           'vfork']),
 'sched_setattr': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioperm',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'migrate_pages',
                       'move_pages',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'msgrcv',
                       'perf_event_open',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'rt_sigaction',
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getattr',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setattr',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'sigaction',
                       'sigaltstack',
                       'signal',
                       'umount',
                       'vfork']),
 'select': set(['capget',
                'clock_nanosleep',
                'clone',
                'dup2',
                'dup3',
                'epoll_wait',
                'faccessat',
                'fchmod',
                'fchmodat',
                'fchown',
```

```
8970                 'fchownat',
8971                 'fork',
8972                 'fstat',
8973                 'ftruncate',
8974                 'futex',
8975                 'futimesat',
8976                 'get_robust_list',
8977                 'getitimer',
8978                 'getpgid',
8979                 'getppid',
8980                 'getpriority',
8981                 'getrusage',
8982                 'getsid',
8983                 'inotify_add_watch',
8984                 'io_getevents',
8985                 'ioctl',
8986                 'ioprio_get',
8987                 'ioprio_set',
8988                 'keyctl',
8989                 'kill',
8990                 'linkat',
8991                 'memfd_create',
8992                 'migrate_pages',
8993                 'move_pages',
8994                 'mq_getsetattr',
8995                 'mq_notify',
8996                 'mq_timedreceive',
8997                 'mq_timedsend',
8998                 'mq_unlink',
8999                 'msgrcv',
9000                 'nanosleep',
9001                 'newfstat',
9002                 'perf_event_open',
9003                 'personality',
9004                 'poll',
9005                 'ppoll',
9006                 'prctl',
9007                 'prlimit64',
9008                 'pselect6',
9009                 'ptrace',
9010                 'readlinkat',
9011                 'recvmmsg',
9012                 'rt_sigaction',
9013                 'rt_sigprocmask',
9014                 'rt_sigtimedwait',
9015                 'sched_getaffinity',
9016                 'sched_getattr',
9017                 'sched_getparam',
9018                 'sched_getscheduler',
9019                 'sched_rr_get_interval',
9020                 'sched_setaffinity',
9021                 'sched_setattr',
9022                 'sched_setparam',
9023                 'sched_setscheduler',
9024                 'select',
9025                 'semtimedop',
9026                 'sendfile',
9027                 'sendfile64',
9028                 'setitimer',
9029                 'setns',
9030                 'setpgid',
9031                 'setpriority',
9032                 'setsid',
9033                 'settimeofday',
9034                 'sigaction',
9035                 'sigaltstack',
```



```
9036                 'signal',
9037                 'stime',
9038                 'swapoff',
9039                 'swapon',
9040                 'timer_gettime',
9041                 'timer_settime',
9042                 'timerfd_gettime',
9043                 'timerfd_settime',
9044                 'umount',
9045                 'unlink',
9046                 'unlinkat',
9047                 'unshare',
9048                 'uselib',
9049                 'utime',
9050                 'vfork']),
9051  'semctl': set(['semctl', 'semtimedop']),
9052  'semtimedop': set(['accept4',
9053                     'acct',
9054                     'bpf',
9055                     'brk',
9056                     'capget',
9057                     'clock_nanosleep',
9058                     'clone',
9059                     'delete_module',
9060                     'dup',
9061                     'dup2',
9062                     'dup3',
9063                     'epoll_create1',
9064                     'epoll_ctl',
9065                     'epoll_wait',
9066                     'eventfd2',
9067                     'exit_group',
9068                     'faccessat',
9069                     'fchmod',
9070                     'fchmodat',
9071                     'fchown',
9072                     'fchownat',
9073                     'finit_module',
9074                     'flock',
9075                     'fork',
9076                     'fstat',
9077                     'ftruncate',
9078                     'futex',
9079                     'futimesat',
9080                     'get_curr_temp',
9081                     'get_mempolicy',
9082                     'get_robust_list',
9083                     'get_trip_temp',
9084                     'getcwd',
9085                     'getgroups',
9086                     'getgroups16',
9087                     'getitimer',
9088                     'getpgid',
9089                     'getppid',
9090                     'getpriority',
9091                     'getrusage',
9092                     'getsid',
9093                     'init_module',
9094                     'inotify_add_watch',
9095                     'inotify_init1',
9096                     'inotify_rm_watch',
9097                     'io_cancel',
9098                     'io_destroy',
9099                     'io_getevents',
9100                     'io_setup',
9101                     'io_submit',
```

```
                    'ioctl',
                    'ioprio_get',
                    'ioprio_set',
                    'kexec_load',
                    'keyctl',
                    'kill',
                    'linkat',
                    'lookup_dcookie',
                    'madvise',
                    'mbind',
                    'memfd_create',
                    'migrate_pages',
                    'mincore',
                    'mkdirat',
                    'mknodat',
                    'mlockall',
                    'mmap_pgoff',
                    'modify_ldt',
                    'move_pages',
                    'mprotect',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'mq_unlink',
                    'mremap',
                    'msgctl',
                    'msgrcv',
                    'msgsnd',
                    'munlock',
                    'munlockall',
                    'nanosleep',
                    'newfstat',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pivot_root',
                    'pkey_mprotect',
                    'poll',
                    'ppoll',
                    'prctl',
                    'prlimit64',
                    'pselect6',
                    'ptrace',
                    'quotactl',
                    'readahead',
                    'readlinkat',
                    'reboot',
                    'recvmmsg',
                    'remap_file_pages',
                    'renameat2',
                    'request_key',
                    'rmdir',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
```



```
                    'sched_setparam',
                    'sched_setscheduler',
                    'sched_yield',
                    'select',
                    'semctl',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'set_trip_temp',
                    'setgid',
                    'setgroups',
                    'setgroups16',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setregid',
                    'setresgid',
                    'setresuid',
                    'setreuid',
                    'setsid',
                    'settimeofday',
                    'setuid',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'socket',
                    'socketpair',
                    'splice',
                    'stime',
                    'swapoff',
                    'swapon',
                    'symlinkat',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'timer_create',
                    'timer_delete',
                    'timer_getoverrun',
                    'timer_gettime',
                    'timer_settime',
                    'timerfd_create',
                    'timerfd_gettime',
                    'timerfd_settime',
                    'umount',
                    'unlink',
                    'unlinkat',
                    'unshare',
                    'uselib',
                    'ustat',
                    'utime',
                    'vfork',
                    'vmsplice']),
    'sendfile': set(['accept4',
                     'acct',
                     'bind',
                     'bpf',
                     'connect',
                     'copy_file_range',
                     'dup',
                     'dup3',
                     'epoll_create1',
                     'epoll_ctl',
```

```
9234                    'epoll_wait',
9235                    'eventfd2',
9236                    'fallocate',
9237                    'fchdir',
9238                    'fchmod',
9239                    'fchown',
9240                    'fcntl',
9241                    'fcntl64',
9242                    'fdatasync',
9243                    'fgetxattr',
9244                    'flistxattr',
9245                    'flock',
9246                    'fremovexattr',
9247                    'fsetxattr',
9248                    'fstatfs',
9249                    'fstatfs64',
9250                    'fsync',
9251                    'ftruncate',
9252                    'futimesat',
9253                    'getdents',
9254                    'getdents64',
9255                    'getpeername',
9256                    'getsockname',
9257                    'getsockopt',
9258                    'inotify_add_watch',
9259                    'inotify_rm_watch',
9260                    'ioctl',
9261                    'listen',
9262                    'llseek',
9263                    'lseek',
9264                    'memfd_create',
9265                    'mmap_pgoff',
9266                    'mq_getsetattr',
9267                    'mq_notify',
9268                    'mq_open',
9269                    'mq_timedreceive',
9270                    'mq_timedsend',
9271                    'old_readdir',
9272                    'open',
9273                    'open_by_handle_at',
9274                    'openat',
9275                    'perf_event_open',
9276                    'pipe2',
9277                    'pread64',
9278                    'preadv',
9279                    'preadv2',
9280                    'preadv64',
9281                    'preadv64v2',
9282                    'pwrite64',
9283                    'pwritev',
9284                    'pwritev2',
9285                    'pwritev64',
9286                    'pwritev64v2',
9287                    'read',
9288                    'readahead',
9289                    'readv',
9290                    'recvfrom',
9291                    'remap_file_pages',
9292                    'sendfile',
9293                    'sendfile64',
9294                    'sendto',
9295                    'setns',
9296                    'setsockopt',
9297                    'shmat',
9298                    'shmctl',
9299                    'shmdt',
```



```
9300                    'shutdown',
9301                    'signalfd4',
9302                    'socket',
9303                    'socketpair',
9304                    'splice',
9305                    'swapoff',
9306                    'swapon',
9307                    'sync_file_range',
9308                    'syncfs',
9309                    'tee',
9310                    'uselib',
9311                    'utime',
9312                    'utimensat',
9313                    'vmsplice',
9314                    'write',
9315                    'writev']),
9316  'sendfile64': set(['accept4',
9317                     'acct',
9318                     'bind',
9319                     'bpf',
9320                     'connect',
9321                     'copy_file_range',
9322                     'dup',
9323                     'dup3',
9324                     'epoll_create1',
9325                     'epoll_ctl',
9326                     'epoll_wait',
9327                     'eventfd2',
9328                     'fallocate',
9329                     'fchdir',
9330                     'fchmod',
9331                     'fchown',
9332                     'fcntl',
9333                     'fcntl64',
9334                     'fdatasync',
9335                     'fgetxattr',
9336                     'flistxattr',
9337                     'flock',
9338                     'fremovexattr',
9339                     'fsetxattr',
9340                     'fstatfs',
9341                     'fstatfs64',
9342                     'fsync',
9343                     'ftruncate',
9344                     'futimesat',
9345                     'getdents',
9346                     'getdents64',
9347                     'getpeername',
9348                     'getsockname',
9349                     'getsockopt',
9350                     'inotify_add_watch',
9351                     'inotify_rm_watch',
9352                     'ioctl',
9353                     'listen',
9354                     'llseek',
9355                     'lseek',
9356                     'memfd_create',
9357                     'mmap_pgoff',
9358                     'mq_getsetattr',
9359                     'mq_notify',
9360                     'mq_open',
9361                     'mq_timedreceive',
9362                     'mq_timedsend',
9363                     'old_readdir',
9364                     'open',
9365                     'open_by_handle_at',
```

```
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setns',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'sendmmsg': set(['accept4',
                  'bind',
                  'connect',
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'listen',
                  'recvfrom',
                  'recvmmsg',
                  'recvmsg',
                  'sendmmsg',
                  'sendmsg',
                  'sendto',
                  'setsockopt',
                  'shutdown']),
 'sendmsg': set(['accept4',
                 'bind',
                 'connect',
                 'getpeername',
                 'getsockname',
                 'getsockopt',
                 'listen',
                 'recvfrom',
                 'recvmmsg',
```



```
                 'recvmsg',
                 'sendmmsg',
                 'sendmsg',
                 'sendto',
                 'setsockopt',
                 'shutdown']),
 'sendto': set(['accept4',
                'acct',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
```

```
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'set_mempolicy': set(['get_mempolicy', 'mbind', 'set_mempolicy']),
 'set_thread_area': set(['arch_prctl',
                         'capget',
                         'clone',
                         'fork',
                         'get_robust_list',
                         'getitimer',
                         'getpgid',
                         'getppid',
                         'getpriority',
                         'getrusage',
                         'getsid',
                         'ioperm',
                         'ioprio_get',
                         'ioprio_set',
                         'keyctl',
                         'kill',
                         'migrate_pages',
                         'move_pages',
                         'mq_timedreceive',
                         'mq_timedsend',
                         'msgrcv',
                         'perf_event_open',
                         'prctl',
                         'prlimit64',
                         'ptrace',
                         'rt_sigaction',
                         'rt_sigprocmask',
                         'rt_sigtimedwait',
                         'sched_getaffinity',
                         'sched_getattr',
                         'sched_getparam',
                         'sched_getscheduler',
                         'sched_rr_get_interval',
```



```
                         'sched_setaffinity',
                         'sched_setattr',
                         'sched_setparam',
                         'sched_setscheduler',
                         'semtimedop',
                         'setitimer',
                         'setns',
                         'setpgid',
                         'setpriority',
                         'setsid',
                         'sigaction',
                         'sigaltstack',
                         'signal',
                         'umount',
                         'vfork']),
 'set_trip_temp': set(['get_curr_temp', 'get_trip_temp', 'set_trip_temp']),
 'setdomainname': set(['setns']),
 'setfsgid': set(['capget',
                  'capset',
                  'clone',
                  'epoll_create1',
                  'faccessat',
                  'fork',
                  'get_robust_list',
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsgid',
                  'setfsuid',
                  'setgid',
                  'setitimer',
```

```
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setregid',
                  'setresgid',
                  'setresuid',
                  'setreuid',
                  'setsid',
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setfsuid': set(['capget',
                  'capset',
                  'clone',
                  'epoll_create1',
                  'faccessat',
                  'fork',
                  'get_robust_list',
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsgid',
                  'setfsuid',
                  'setgid',
                  'setitimer',
                  'setns',
                  'setpgid',
```



```
                  'setpriority',
                  'setregid',
                  'setresgid',
                  'setresuid',
                  'setreuid',
                  'setsid',
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setgid': set(['capget',
                'capset',
                'clone',
                'epoll_create1',
                'faccessat',
                'fork',
                'get_robust_list',
                'getgroups',
                'getgroups16',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getresgid',
                'getresgid16',
                'getresuid',
                'getresuid16',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setfsgid',
                'setfsuid',
                'setgid',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setregid',
```

```
                'setresgid',
                'setresuid',
                'setreuid',
                'setsid',
                'setuid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'unshare',
                'vfork']),
 'setgroups16': set(['setgroups', 'setgroups16']),
 'sethostname': set(['setns']),
 'setitimer': set(['adjtimex',
                   'alarm',
                   'clock_adjtime',
                   'exit_group',
                   'getitimer',
                   'getrusage',
                   'ppoll',
                   'select',
                   'setitimer',
                   'settimeofday',
                   'timer_create',
                   'wait4',
                   'waitid']),
 'setns': set(['setns']),
 'setpgid': set(['capget',
                 'clone',
                 'exit_group',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'migrate_pages',
                 'move_pages',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'msgrcv',
                 'perf_event_open',
                 'prctl',
                 'prlimit64',
                 'ptrace',
                 'rt_sigaction',
                 'rt_sigprocmask',
                 'rt_sigtimedwait',
                 'sched_getaffinity',
                 'sched_getattr',
                 'sched_getparam',
                 'sched_getscheduler',
                 'sched_rr_get_interval',
                 'sched_setaffinity',
                 'sched_setattr',
                 'sched_setparam',
                 'sched_setscheduler',
                 'semtimedop',
                 'setitimer',
                 'setns',
```



```
                 'setpgid',
                 'setpriority',
                 'setresuid',
                 'setreuid',
                 'setsid',
                 'setuid',
                 'sigaction',
                 'sigaltstack',
                 'signal',
                 'timer_create',
                 'umount',
                 'vfork']),
 'setpriority': set(['capget',
                     'clone',
                     'fork',
                     'get_robust_list',
                     'getitimer',
                     'getpgid',
                     'getppid',
                     'getpriority',
                     'getrusage',
                     'getsid',
                     'ioprio_get',
                     'ioprio_set',
                     'keyctl',
                     'kill',
                     'migrate_pages',
                     'move_pages',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'msgrcv',
                     'perf_event_open',
                     'prctl',
                     'prlimit64',
                     'ptrace',
                     'rt_sigaction',
                     'rt_sigprocmask',
                     'rt_sigtimedwait',
                     'sched_getaffinity',
                     'sched_getattr',
                     'sched_getparam',
                     'sched_getscheduler',
                     'sched_rr_get_interval',
                     'sched_setaffinity',
                     'sched_setattr',
                     'sched_setparam',
                     'sched_setscheduler',
                     'semtimedop',
                     'setitimer',
                     'setns',
                     'setpgid',
                     'setpriority',
                     'setresuid',
                     'setreuid',
                     'setsid',
                     'setuid',
                     'sigaction',
                     'sigaltstack',
                     'signal',
                     'umount',
                     'vfork']),
 'setregid': set(['capget',
                  'capset',
                  'clone',
                  'epoll_create1',
                  'faccessat',
```

```
                  'fork',
                  'get_robust_list',
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsgid',
                  'setfsuid',
                  'setgid',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setregid',
                  'setresgid',
                  'setresuid',
                  'setreuid',
                  'setsid',
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setresgid': set(['capget',
                   'capset',
                   'clone',
                   'epoll_create1',
                   'faccessat',
                   'fork',
                   'get_robust_list',
```



```
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsgid',
                  'setfsuid',
                   'setgid',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setregid',
                   'setresgid',
                   'setresuid',
                   'setreuid',
                   'setsid',
                   'setuid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'unshare',
                   'vfork']),
 'setresuid': set(['capget',
                   'capset',
                   'clone',
                   'epoll_create1',
                   'faccessat',
                   'fork',
                   'get_robust_list',
                   'getgroups',
                   'getgroups16',
```

```
                    'getitimer',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getresgid',
                    'getresgid16',
                    'getresuid',
                    'getresuid16',
                    'getrusage',
                    'getsid',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setfsgid',
                    'setfsuid',
                    'setgid',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setregid',
                    'setresgid',
                    'setresuid',
                    'setreuid',
                    'setsid',
                    'setuid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'unshare',
                    'vfork']),
 'setreuid': set(['capget',
                  'capset',
                  'clone',
                  'epoll_create1',
                  'faccessat',
                  'fork',
                  'get_robust_list',
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
```



```
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsgid',
                  'setfsuid',
                  'setgid',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setregid',
                  'setresgid',
                  'setresuid',
                  'setreuid',
                  'setsid',
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setrlimit': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrlimit',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
```

```
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'old_getrlimit',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setrlimit',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'setsid': set(['exit_group', 'setsid', 'timer_create']),
 'setsockopt': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
```



```
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'settimeofday': set(['adjtimex',
                      'alarm',
                      'clock_adjtime',
                      'getitimer',
                      'getrusage',
                      'ppoll',
                      'select',
                      'setitimer',
                      'settimeofday',
                      'wait4',
                      'waitid']),
 'setuid': set(['capget',
                'capset',
                'clone',
                'epoll_create1',
                'faccessat',
                'fork',
                'get_robust_list',
                'getgroups',
                'getgroups16',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getresgid',
```

```
10290                 'getresgid16',
10291                 'getresuid',
10292                 'getresuid16',
10293                 'getrusage',
10294                 'getsid',
10295                 'ioprio_get',
10296                 'ioprio_set',
10297                 'keyctl',
10298                 'kill',
10299                 'migrate_pages',
10300                 'move_pages',
10301                 'mq_timedreceive',
10302                 'mq_timedsend',
10303                 'msgrcv',
10304                 'perf_event_open',
10305                 'prctl',
10306                 'prlimit64',
10307                 'ptrace',
10308                 'rt_sigaction',
10309                 'rt_sigprocmask',
10310                 'rt_sigtimedwait',
10311                 'sched_getaffinity',
10312                 'sched_getattr',
10313                 'sched_getparam',
10314                 'sched_getscheduler',
10315                 'sched_rr_get_interval',
10316                 'sched_setaffinity',
10317                 'sched_setattr',
10318                 'sched_setparam',
10319                 'sched_setscheduler',
10320                 'semtimedop',
10321                 'setfsgid',
10322                 'setfsuid',
10323                 'setgid',
10324                 'setitimer',
10325                 'setns',
10326                 'setpgid',
10327                 'setpriority',
10328                 'setregid',
10329                 'setresgid',
10330                 'setresuid',
10331                 'setreuid',
10332                 'setsid',
10333                 'setuid',
10334                 'sigaction',
10335                 'sigaltstack',
10336                 'signal',
10337                 'umount',
10338                 'unshare',
10339                 'vfork']),
10340  'setxattr': set(['accept4',
10341                   'acct',
10342                   'dup',
10343                   'dup3',
10344                   'epoll_create1',
10345                   'epoll_ctl',
10346                   'eventfd2',
10347                   'flock',
10348                   'getcwd',
10349                   'lookup_dcookie',
10350                   'memfd_create',
10351                   'mmap_pgoff',
10352                   'mq_open',
10353                   'open',
10354                   'open_by_handle_at',
10355                   'openat',
```



```
10356                   'perf_event_open',
10357                   'pipe2',
10358                   'pivot_root',
10359                   'quotactl',
10360                   'remap_file_pages',
10361                   'setns',
10362                   'shmat',
10363                   'shmctl',
10364                   'shmdt',
10365                   'socket',
10366                   'socketpair',
10367                   'swapoff',
10368                   'swapon',
10369                   'unshare',
10370                   'uselib']),
10371  'shmat': set(['accept4',
10372                'acct',
10373                'capget',
10374                'clone',
10375                'dup',
10376                'dup3',
10377                'epoll_create1',
10378                'epoll_ctl',
10379                'eventfd2',
10380                'flock',
10381                'fork',
10382                'get_robust_list',
10383                'getcwd',
10384                'getitimer',
10385                'getpgid',
10386                'getppid',
10387                'getpriority',
10388                'getrusage',
10389                'getsid',
10390                'ioprio_get',
10391                'ioprio_set',
10392                'keyctl',
10393                'kill',
10394                'lookup_dcookie',
10395                'memfd_create',
10396                'migrate_pages',
10397                'mmap_pgoff',
10398                'move_pages',
10399                'mq_open',
10400                'mq_timedreceive',
10401                'mq_timedsend',
10402                'msgrcv',
10403                'open',
10404                'open_by_handle_at',
10405                'openat',
10406                'perf_event_open',
10407                'pipe2',
10408                'pivot_root',
10409                'prctl',
10410                'prlimit64',
10411                'ptrace',
10412                'quotactl',
10413                'remap_file_pages',
10414                'rt_sigaction',
10415                'rt_sigprocmask',
10416                'rt_sigtimedwait',
10417                'sched_getaffinity',
10418                'sched_getattr',
10419                'sched_getparam',
10420                'sched_getscheduler',
10421                'sched_rr_get_interval',
```

```
10422                 'sched_setaffinity',
10423                 'sched_setattr',
10424                 'sched_setparam',
10425                 'sched_setscheduler',
10426                 'semtimedop',
10427                 'setitimer',
10428                 'setns',
10429                 'setpgid',
10430                 'setpriority',
10431                 'setsid',
10432                 'shmat',
10433                 'shmctl',
10434                 'shmdt',
10435                 'sigaction',
10436                 'sigaltstack',
10437                 'signal',
10438                 'socket',
10439                 'socketpair',
10440                 'swapoff',
10441                 'swapon',
10442                 'umount',
10443                 'unshare',
10444                 'uselib',
10445                 'vfork']),
10446  'shmctl': set(['accept4',
10447                 'acct',
10448                 'capget',
10449                 'clone',
10450                 'dup',
10451                 'dup3',
10452                 'epoll_create1',
10453                 'epoll_ctl',
10454                 'eventfd2',
10455                 'flock',
10456                 'fork',
10457                 'get_robust_list',
10458                 'getitimer',
10459                 'getpgid',
10460                 'getppid',
10461                 'getpriority',
10462                 'getrusage',
10463                 'getsid',
10464                 'ioperm',
10465                 'ioprio_get',
10466                 'ioprio_set',
10467                 'keyctl',
10468                 'kill',
10469                 'memfd_create',
10470                 'migrate_pages',
10471                 'mmap_pgoff',
10472                 'move_pages',
10473                 'mq_open',
10474                 'mq_timedreceive',
10475                 'mq_timedsend',
10476                 'mq_unlink',
10477                 'msgctl',
10478                 'msgget',
10479                 'msgrcv',
10480                 'msgsnd',
10481                 'open',
10482                 'open_by_handle_at',
10483                 'openat',
10484                 'perf_event_open',
10485                 'pipe2',
10486                 'prctl',
10487                 'prlimit64',
```



```
10488                 'ptrace',
10489                 'remap_file_pages',
10490                 'rt_sigaction',
10491                 'rt_sigprocmask',
10492                 'rt_sigtimedwait',
10493                 'sched_getaffinity',
10494                 'sched_getattr',
10495                 'sched_getparam',
10496                 'sched_getscheduler',
10497                 'sched_rr_get_interval',
10498                 'sched_setaffinity',
10499                 'sched_setattr',
10500                 'sched_setparam',
10501                 'sched_setscheduler',
10502                 'semctl',
10503                 'semget',
10504                 'semtimedop',
10505                 'setitimer',
10506                 'setns',
10507                 'setpgid',
10508                 'setpriority',
10509                 'setsid',
10510                 'shmat',
10511                 'shmctl',
10512                 'shmdt',
10513                 'shmget',
10514                 'sigaction',
10515                 'sigaltstack',
10516                 'signal',
10517                 'socket',
10518                 'socketpair',
10519                 'swapoff',
10520                 'swapon',
10521                 'umount',
10522                 'uselib',
10523                 'vfork']),
10524  'shmdt': set(['brk',
10525                'get_mempolicy',
10526                'madvise',
10527                'mincore',
10528                'mlockall',
10529                'mprotect',
10530                'mremap',
10531                'munlock',
10532                'munlockall',
10533                'pkey_mprotect',
10534                'prctl',
10535                'remap_file_pages',
10536                'shmdt']),
10537  'shutdown': set(['accept4',
10538                   'bind',
10539                   'bpf',
10540                   'connect',
10541                   'copy_file_range',
10542                   'epoll_ctl',
10543                   'epoll_wait',
10544                   'fallocate',
10545                   'fchdir',
10546                   'fchmod',
10547                   'fchown',
10548                   'fcntl',
10549                   'fcntl64',
10550                   'fdatasync',
10551                   'fgetxattr',
10552                   'flistxattr',
10553                   'flock',
```

```
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
  'sigaction': set(['capget',
                    'clone',
                    'fork',
                    'get_robust_list',
                    'getitimer',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'ioperm',
                    'ioprio_get',
                    'ioprio_set',
```



```
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
  'signalfd4': set(['accept4',
                    'acct',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'epoll_wait',
                    'eventfd2',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
```

```
10686                   'getsockname',
10687                   'getsockopt',
10688                   'inotify_add_watch',
10689                   'inotify_rm_watch',
10690                   'ioctl',
10691                   'listen',
10692                   'llseek',
10693                   'lseek',
10694                   'memfd_create',
10695                   'mmap_pgoff',
10696                   'mq_getsetattr',
10697                   'mq_notify',
10698                   'mq_open',
10699                   'mq_timedreceive',
10700                   'mq_timedsend',
10701                   'old_readdir',
10702                   'open',
10703                   'open_by_handle_at',
10704                   'openat',
10705                   'perf_event_open',
10706                   'pipe2',
10707                   'pread64',
10708                   'preadv',
10709                   'preadv2',
10710                   'preadv64',
10711                   'preadv64v2',
10712                   'pwrite64',
10713                   'pwritev',
10714                   'pwritev2',
10715                   'pwritev64',
10716                   'pwritev64v2',
10717                   'read',
10718                   'readahead',
10719                   'readv',
10720                   'recvfrom',
10721                   'remap_file_pages',
10722                   'sendfile',
10723                   'sendfile64',
10724                   'sendto',
10725                   'setns',
10726                   'setsockopt',
10727                   'shmat',
10728                   'shmctl',
10729                   'shmdt',
10730                   'shutdown',
10731                   'signalfd4',
10732                   'socket',
10733                   'socketpair',
10734                   'splice',
10735                   'swapoff',
10736                   'swapon',
10737                   'sync_file_range',
10738                   'syncfs',
10739                   'tee',
10740                   'uselib',
10741                   'utime',
10742                   'utimensat',
10743                   'vmsplice',
10744                   'write',
10745                   'writev']),
10746   'splice': set(['accept4',
10747                  'acct',
10748                  'bind',
10749                  'bpf',
10750                  'connect',
10751                  'copy_file_range',
```



```
10752                  'dup',
10753                  'dup3',
10754                  'epoll_create1',
10755                  'epoll_ctl',
10756                  'epoll_wait',
10757                  'eventfd2',
10758                  'fallocate',
10759                  'fchdir',
10760                  'fchmod',
10761                  'fchown',
10762                  'fcntl',
10763                  'fcntl64',
10764                  'fdatasync',
10765                  'fgetxattr',
10766                  'flistxattr',
10767                  'flock',
10768                  'fremovexattr',
10769                  'fsetxattr',
10770                  'fstatfs',
10771                  'fstatfs64',
10772                  'fsync',
10773                  'ftruncate',
10774                  'futimesat',
10775                  'getdents',
10776                  'getdents64',
10777                  'getpeername',
10778                  'getsockname',
10779                  'getsockopt',
10780                  'inotify_add_watch',
10781                  'inotify_rm_watch',
10782                  'ioctl',
10783                  'listen',
10784                  'llseek',
10785                  'lseek',
10786                  'memfd_create',
10787                  'mmap_pgoff',
10788                  'mq_getsetattr',
10789                  'mq_notify',
10790                  'mq_open',
10791                  'mq_timedreceive',
10792                  'mq_timedsend',
10793                  'old_readdir',
10794                  'open',
10795                  'open_by_handle_at',
10796                  'openat',
10797                  'perf_event_open',
10798                  'pipe2',
10799                  'pread64',
10800                  'preadv',
10801                  'preadv2',
10802                  'preadv64',
10803                  'preadv64v2',
10804                  'pwrite64',
10805                  'pwritev',
10806                  'pwritev2',
10807                  'pwritev64',
10808                  'pwritev64v2',
10809                  'read',
10810                  'readahead',
10811                  'readv',
10812                  'recvfrom',
10813                  'remap_file_pages',
10814                  'sendfile',
10815                  'sendfile64',
10816                  'sendto',
10817                  'setns',
```

```
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'stat': set(['fstat', 'lstat', 'newfstat', 'stat']),
 'statfs': set(['fstatfs', 'fstatfs64', 'statfs', 'statfs64', 'ustat']),
 'statfs64': set(['fstatfs', 'fstatfs64', 'statfs', 'statfs64', 'ustat']),
 'swapoff': set(['accept4',
                 'acct',
                 'capget',
                 'clone',
                 'dup',
                 'dup3',
                 'epoll_create1',
                 'epoll_ctl',
                 'eventfd2',
                 'flock',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'memfd_create',
                 'migrate_pages',
                 'mmap_pgoff',
                 'move_pages',
                 'mq_open',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'msgrcv',
                 'open',
                 'open_by_handle_at',
                 'openat',
                 'perf_event_open',
                 'pipe2',
                 'prctl',
                 'prlimit64',
                 'ptrace',
                 'remap_file_pages',
                 'rt_sigaction',
                 'rt_sigprocmask',
                 'rt_sigtimedwait',
                 'sched_getaffinity',
```



```
                 'sched_getattr',
                 'sched_getparam',
                 'sched_getscheduler',
                 'sched_rr_get_interval',
                 'sched_setaffinity',
                 'sched_setattr',
                 'sched_setparam',
                 'sched_setscheduler',
                 'semtimedop',
                 'setitimer',
                 'setns',
                 'setpgid',
                 'setpriority',
                 'setsid',
                 'shmat',
                 'shmctl',
                 'shmdt',
                 'sigaction',
                 'sigaltstack',
                 'signal',
                 'socket',
                 'socketpair',
                 'swapoff',
                 'swapon',
                 'umount',
                 'uselib',
                 'vfork']),
 'swapon': set(['faccessat',
                'fchmod',
                'fchmodat',
                'fchown',
                'fchownat',
                'ftruncate',
                'inotify_add_watch',
                'ioctl',
                'linkat',
                'memfd_create',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'readlinkat',
                'sendfile',
                'sendfile64',
                'swapoff',
                'swapon',
                'unlink',
                'unlinkat',
                'uselib']),
 'symlinkat': set(['acct',
                   'mq_open',
                   'mq_unlink',
                   'open',
                   'openat',
                   'quotactl',
                   'renameat2',
                   'rmdir',
                   'swapoff',
                   'swapon',
                   'symlinkat',
                   'sysfs',
                   'unlink',
                   'unlinkat',
                   'uselib']),
 'sync_file_range': set(['accept4',
```

```
10950                         'bind',
10951                         'bpf',
10952                         'connect',
10953                         'copy_file_range',
10954                         'epoll_ctl',
10955                         'epoll_wait',
10956                         'fallocate',
10957                         'fchdir',
10958                         'fchmod',
10959                         'fchown',
10960                         'fcntl',
10961                         'fcntl64',
10962                         'fdatasync',
10963                         'fgetxattr',
10964                         'flistxattr',
10965                         'flock',
10966                         'fremovexattr',
10967                         'fsetxattr',
10968                         'fstatfs',
10969                         'fstatfs64',
10970                         'fsync',
10971                         'ftruncate',
10972                         'futimesat',
10973                         'getdents',
10974                         'getdents64',
10975                         'getpeername',
10976                         'getsockname',
10977                         'getsockopt',
10978                         'inotify_add_watch',
10979                         'inotify_rm_watch',
10980                         'ioctl',
10981                         'listen',
10982                         'llseek',
10983                         'lseek',
10984                         'mq_getsetattr',
10985                         'mq_notify',
10986                         'mq_timedreceive',
10987                         'mq_timedsend',
10988                         'old_readdir',
10989                         'perf_event_open',
10990                         'pread64',
10991                         'preadv',
10992                         'preadv2',
10993                         'preadv64',
10994                         'preadv64v2',
10995                         'pwrite64',
10996                         'pwritev',
10997                         'pwritev2',
10998                         'pwritev64',
10999                         'pwritev64v2',
11000                         'read',
11001                         'readahead',
11002                         'readv',
11003                         'recvfrom',
11004                         'sendfile',
11005                         'sendfile64',
11006                         'sendto',
11007                         'setsockopt',
11008                         'shutdown',
11009                         'signalfd4',
11010                         'splice',
11011                         'sync_file_range',
11012                         'syncfs',
11013                         'tee',
11014                         'utime',
11015                         'utimensat',
```



```
11016                         'vmsplice',
11017                         'write',
11018                         'writev']),
11019  'syncfs': set(['accept4',
11020                 'bind',
11021                 'bpf',
11022                 'connect',
11023                 'copy_file_range',
11024                 'epoll_ctl',
11025                 'epoll_wait',
11026                 'fallocate',
11027                 'fchdir',
11028                 'fchmod',
11029                 'fchown',
11030                 'fcntl',
11031                 'fcntl64',
11032                 'fdatasync',
11033                 'fgetxattr',
11034                 'flistxattr',
11035                 'flock',
11036                 'fremovexattr',
11037                 'fsetxattr',
11038                 'fstatfs',
11039                 'fstatfs64',
11040                 'fsync',
11041                 'ftruncate',
11042                 'futimesat',
11043                 'getdents',
11044                 'getdents64',
11045                 'getpeername',
11046                 'getsockname',
11047                 'getsockopt',
11048                 'inotify_add_watch',
11049                 'inotify_rm_watch',
11050                 'ioctl',
11051                 'listen',
11052                 'llseek',
11053                 'lseek',
11054                 'mq_getsetattr',
11055                 'mq_notify',
11056                 'mq_timedreceive',
11057                 'mq_timedsend',
11058                 'old_readdir',
11059                 'perf_event_open',
11060                 'pread64',
11061                 'preadv',
11062                 'preadv2',
11063                 'preadv64',
11064                 'preadv64v2',
11065                 'pwrite64',
11066                 'pwritev',
11067                 'pwritev2',
11068                 'pwritev64',
11069                 'pwritev64v2',
11070                 'read',
11071                 'readahead',
11072                 'readv',
11073                 'recvfrom',
11074                 'sendfile',
11075                 'sendfile64',
11076                 'sendto',
11077                 'setsockopt',
11078                 'shutdown',
11079                 'signalfd4',
11080                 'splice',
11081                 'sync_file_range',
```

```
11082                 'syncfs',
11083                 'tee',
11084                 'utime',
11085                 'utimensat',
11086                 'vmsplice',
11087                 'write',
11088                 'writev']),
11089  'sysctl': set(['sysctl']),
11090  'sysfs': set(['acct',
11091                'mq_open',
11092                'mq_unlink',
11093                'open',
11094                'openat',
11095                'quotactl',
11096                'renameat2',
11097                'rmdir',
11098                'swapoff',
11099                'swapon',
11100                'symlinkat',
11101                'sysfs',
11102                'unlink',
11103                'unlinkat',
11104                'uselib']),
11105  'sysinfo': set(['capget',
11106                  'clock_nanosleep',
11107                  'clone',
11108                  'epoll_wait',
11109                  'faccessat',
11110                  'fchmod',
11111                  'fchmodat',
11112                  'fchown',
11113                  'fchownat',
11114                  'fork',
11115                  'fstat',
11116                  'ftruncate',
11117                  'futex',
11118                  'futimesat',
11119                  'get_robust_list',
11120                  'getitimer',
11121                  'getpgid',
11122                  'getppid',
11123                  'getpriority',
11124                  'getrusage',
11125                  'getsid',
11126                  'inotify_add_watch',
11127                  'io_getevents',
11128                  'ioctl',
11129                  'ioperm',
11130                  'ioprio_get',
11131                  'ioprio_set',
11132                  'keyctl',
11133                  'kill',
11134                  'linkat',
11135                  'memfd_create',
11136                  'migrate_pages',
11137                  'move_pages',
11138                  'mq_getsetattr',
11139                  'mq_notify',
11140                  'mq_timedreceive',
11141                  'mq_timedsend',
11142                  'mq_unlink',
11143                  'msgrcv',
11144                  'nanosleep',
11145                  'newfstat',
11146                  'perf_event_open',
11147                  'poll',
```



```
11148                 'ppoll',
11149                 'prctl',
11150                 'prlimit64',
11151                 'pselect6',
11152                 'ptrace',
11153                 'readlinkat',
11154                 'recvmmsg',
11155                 'rt_sigaction',
11156                 'rt_sigprocmask',
11157                 'rt_sigtimedwait',
11158                 'sched_getaffinity',
11159                 'sched_getattr',
11160                 'sched_getparam',
11161                 'sched_getscheduler',
11162                 'sched_rr_get_interval',
11163                 'sched_setaffinity',
11164                 'sched_setattr',
11165                 'sched_setparam',
11166                 'sched_setscheduler',
11167                 'select',
11168                 'semtimedop',
11169                 'sendfile',
11170                 'sendfile64',
11171                 'setitimer',
11172                 'setns',
11173                 'setpgid',
11174                 'setpriority',
11175                 'setsid',
11176                 'settimeofday',
11177                 'sigaction',
11178                 'sigaltstack',
11179                 'signal',
11180                 'stime',
11181                 'swapoff',
11182                 'swapon',
11183                 'sysinfo',
11184                 'timer_gettime',
11185                 'timer_settime',
11186                 'timerfd_gettime',
11187                 'timerfd_settime',
11188                 'umount',
11189                 'unlink',
11190                 'unlinkat',
11191                 'uselib',
11192                 'utime',
11193                 'vfork']),
11194  'syslog': set(['capget',
11195                 'clone',
11196                 'fork',
11197                 'get_robust_list',
11198                 'getitimer',
11199                 'getpgid',
11200                 'getppid',
11201                 'getpriority',
11202                 'getrusage',
11203                 'getsid',
11204                 'ioperm',
11205                 'ioprio_get',
11206                 'ioprio_set',
11207                 'keyctl',
11208                 'kill',
11209                 'migrate_pages',
11210                 'move_pages',
11211                 'mq_timedreceive',
11212                 'mq_timedsend',
11213                 'msgrcv',
```

```
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'tee': set(['accept4',
             'acct',
             'bind',
             'bpf',
             'connect',
             'copy_file_range',
             'dup',
             'dup3',
             'epoll_create1',
             'epoll_ctl',
             'epoll_wait',
             'eventfd2',
             'fallocate',
             'fchdir',
             'fchmod',
             'fchown',
             'fcntl',
             'fcntl64',
             'fdatasync',
             'fgetxattr',
             'flistxattr',
             'flock',
             'fremovexattr',
             'fsetxattr',
             'fstatfs',
             'fstatfs64',
             'fsync',
             'ftruncate',
             'futimesat',
             'getdents',
             'getdents64',
             'getpeername',
             'getsockname',
             'getsockopt',
             'inotify_add_watch',
             'inotify_rm_watch',
             'ioctl',
             'listen',
             'llseek',
```



```
             'lseek',
             'memfd_create',
             'mmap_pgoff',
             'mq_getsetattr',
             'mq_notify',
             'mq_open',
             'mq_timedreceive',
             'mq_timedsend',
             'old_readdir',
             'open',
             'open_by_handle_at',
             'openat',
             'perf_event_open',
             'pipe2',
             'pread64',
             'preadv',
             'preadv2',
             'preadv64',
             'preadv64v2',
             'pwrite64',
             'pwritev',
             'pwritev2',
             'pwritev64',
             'pwritev64v2',
             'read',
             'readahead',
             'readv',
             'recvfrom',
             'remap_file_pages',
             'sendfile',
             'sendfile64',
             'sendto',
             'setns',
             'setsockopt',
             'shmat',
             'shmctl',
             'shmdt',
             'shutdown',
             'signalfd4',
             'socket',
             'socketpair',
             'splice',
             'swapoff',
             'swapon',
             'sync_file_range',
             'syncfs',
             'tee',
             'uselib',
             'utime',
             'utimensat',
             'vmsplice',
             'write',
             'writev']),
 'tgkill': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
```

```
11346                 'kill',
11347                 'migrate_pages',
11348                 'move_pages',
11349                 'mq_timedreceive',
11350                 'mq_timedsend',
11351                 'msgrcv',
11352                 'perf_event_open',
11353                 'prctl',
11354                 'prlimit64',
11355                 'ptrace',
11356                 'rt_sigaction',
11357                 'rt_sigprocmask',
11358                 'rt_sigtimedwait',
11359                 'sched_getaffinity',
11360                 'sched_getattr',
11361                 'sched_getparam',
11362                 'sched_getscheduler',
11363                 'sched_rr_get_interval',
11364                 'sched_setaffinity',
11365                 'sched_setattr',
11366                 'sched_setparam',
11367                 'sched_setscheduler',
11368                 'semtimedop',
11369                 'setitimer',
11370                 'setns',
11371                 'setpgid',
11372                 'setpriority',
11373                 'setsid',
11374                 'sigaction',
11375                 'sigaltstack',
11376                 'signal',
11377                 'umount',
11378                 'vfork']),
11379  'timer_create': set(['clock_adjtime',
11380                       'clock_getres',
11381                       'clock_gettime',
11382                       'clock_nanosleep',
11383                       'clock_settime',
11384                       'timer_create',
11385                       'timer_delete',
11386                       'timer_getoverrun',
11387                       'timer_gettime',
11388                       'timer_settime']),
11389  'timer_delete': set(['capget',
11390                       'clock_adjtime',
11391                       'clock_getres',
11392                       'clock_gettime',
11393                       'clock_nanosleep',
11394                       'clock_settime',
11395                       'clone',
11396                       'fork',
11397                       'get_robust_list',
11398                       'getitimer',
11399                       'getpgid',
11400                       'getppid',
11401                       'getpriority',
11402                       'getrusage',
11403                       'getsid',
11404                       'ioprio_get',
11405                       'ioprio_set',
11406                       'keyctl',
11407                       'kill',
11408                       'migrate_pages',
11409                       'move_pages',
11410                       'mq_timedreceive',
11411                       'mq_timedsend',
```



```
11412                       'msgrcv',
11413                       'perf_event_open',
11414                       'prctl',
11415                       'prlimit64',
11416                       'ptrace',
11417                       'rt_sigaction',
11418                       'rt_sigprocmask',
11419                       'rt_sigtimedwait',
11420                       'sched_getaffinity',
11421                       'sched_getattr',
11422                       'sched_getparam',
11423                       'sched_getscheduler',
11424                       'sched_rr_get_interval',
11425                       'sched_setaffinity',
11426                       'sched_setattr',
11427                       'sched_setparam',
11428                       'sched_setscheduler',
11429                       'semtimedop',
11430                       'setitimer',
11431                       'setns',
11432                       'setpgid',
11433                       'setpriority',
11434                       'setsid',
11435                       'sigaction',
11436                       'sigaltstack',
11437                       'signal',
11438                       'timer_create',
11439                       'timer_delete',
11440                       'timer_getoverrun',
11441                       'timer_gettime',
11442                       'timer_settime',
11443                       'umount',
11444                       'vfork']),
11445  'timer_getoverrun': set(['capget',
11446                           'clone',
11447                           'fork',
11448                           'get_robust_list',
11449                           'getitimer',
11450                           'getpgid',
11451                           'getppid',
11452                           'getpriority',
11453                           'getrusage',
11454                           'getsid',
11455                           'ioprio_get',
11456                           'ioprio_set',
11457                           'keyctl',
11458                           'kill',
11459                           'migrate_pages',
11460                           'move_pages',
11461                           'mq_timedreceive',
11462                           'mq_timedsend',
11463                           'msgrcv',
11464                           'perf_event_open',
11465                           'prctl',
11466                           'prlimit64',
11467                           'ptrace',
11468                           'rt_sigaction',
11469                           'rt_sigprocmask',
11470                           'rt_sigtimedwait',
11471                           'sched_getaffinity',
11472                           'sched_getattr',
11473                           'sched_getparam',
11474                           'sched_getscheduler',
11475                           'sched_rr_get_interval',
11476                           'sched_setaffinity',
11477                           'sched_setattr',
```

```
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setsid',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'timer_create',
                        'timer_delete',
                        'timer_getoverrun',
                        'timer_gettime',
                        'timer_settime',
                        'umount',
                        'vfork']),
 'timer_gettime': set(['clock_adjtime',
                       'clock_getres',
                       'clock_gettime',
                       'clock_nanosleep',
                       'clock_settime',
                       'timer_create',
                       'timer_delete',
                       'timer_gettime',
                       'timer_settime']),
 'timer_settime': set(['clock_adjtime',
                       'clock_getres',
                       'clock_gettime',
                       'clock_nanosleep',
                       'clock_settime',
                       'timer_create',
                       'timer_delete',
                       'timer_gettime',
                       'timer_settime']),
 'timerfd_create': set(['timerfd_create',
                        'timerfd_gettime',
                        'timerfd_settime']),
 'timerfd_gettime': set(['timerfd_create',
                         'timerfd_gettime',
                         'timerfd_settime']),
 'timerfd_settime': set(['timerfd_create',
                         'timerfd_gettime',
                         'timerfd_settime']),
 'tkill': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'migrate_pages',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'msgrcv',
               'perf_event_open',
               'prctl',
```



```
               'prlimit64',
               'ptrace',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
               'semtimedop',
               'setitimer',
               'setns',
               'setpgid',
               'setpriority',
               'setsid',
               'sigaction',
               'sigaltstack',
               'signal',
               'umount',
               'vfork']),
 'umount': set(['accept4',
                'acct',
                'capget',
                'clone',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'eventfd2',
                'flock',
                'fork',
                'get_robust_list',
                'getcwd',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'lookup_dcookie',
                'memfd_create',
                'migrate_pages',
                'mmap_pgoff',
                'move_pages',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'msgrcv',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'prctl',
                'prlimit64',
```

```
                'ptrace',
                'quotactl',
                'remap_file_pages',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setresuid',
                'setreuid',
                'setsid',
                'setuid',
                'shmat',
                'shmctl',
                'shmdt',
                'sigaction',
                'sigaltstack',
                'signal',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'syncfs',
                'umount',
                'unshare',
                'uselib',
                'ustat',
                'vfork']),
 'uname': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'migrate_pages',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'msgrcv',
               'perf_event_open',
               'personality',
               'prctl',
               'prlimit64',
               'ptrace',
               'rt_sigaction',
               'rt_sigprocmask',
```



```
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
               'semtimedop',
               'setitimer',
               'setns',
               'setpgid',
               'setpriority',
               'setsid',
               'sigaction',
               'sigaltstack',
               'signal',
               'umount',
               'vfork']),
 'unlink': set(['accept4',
                'acct',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'eventfd2',
                'flock',
                'ftruncate',
                'getcwd',
                'linkat',
                'lookup_dcookie',
                'memfd_create',
                'mkdirat',
                'mknodat',
                'mmap_pgoff',
                'mq_open',
                'mq_unlink',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'quotactl',
                'remap_file_pages',
                'renameat2',
                'rmdir',
                'setns',
                'shmat',
                'shmctl',
                'shmdt',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'symlinkat',
                'unlink',
                'unlinkat',
                'unshare',
                'uselib']),
 'unlinkat': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
```

```
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'ftruncate',
                  'getcwd',
                  'linkat',
                  'lookup_dcookie',
                  'memfd_create',
                  'mkdirat',
                  'mknodat',
                  'mmap_pgoff',
                  'mq_open',
                  'mq_unlink',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pivot_root',
                  'quotactl',
                  'remap_file_pages',
                  'renameat2',
                  'rmdir',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'symlinkat',
                  'unlink',
                  'unlinkat',
                  'unshare',
                  'uselib']),
 'unshare': set(['unshare']),
 'uselib': set(['acct',
                'getcwd',
                'mq_open',
                'mq_unlink',
                'pivot_root',
                'quotactl',
                'swapon',
                'syncfs',
                'umount',
                'uselib',
                'ustat']),
 'ustat': set(['quotactl', 'swapon', 'syncfs', 'umount', 'ustat']),
 'utime': set(['accept4',
               'bind',
               'bpf',
               'clock_nanosleep',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'faccessat',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchmodat',
               'fchown',
               'fchownat',
               'fcntl',
```



```
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstat',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futex',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'io_getevents',
               'ioctl',
               'linkat',
               'listen',
               'llseek',
               'lseek',
               'memfd_create',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'mq_unlink',
               'nanosleep',
               'newfstat',
               'old_readdir',
               'perf_event_open',
               'poll',
               'ppoll',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pselect6',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readlinkat',
               'readv',
               'recvfrom',
               'recvmmsg',
               'rt_sigtimedwait',
               'sched_rr_get_interval',
               'select',
               'semtimedop',
               'sendfile',
               'sendfile64',
               'sendto',
               'setsockopt',
               'settimeofday',
               'shutdown',
```

```
11874                 'signalfd4',
11875                 'splice',
11876                 'stime',
11877                 'swapoff',
11878                 'swapon',
11879                 'sync_file_range',
11880                 'syncfs',
11881                 'tee',
11882                 'timer_gettime',
11883                 'timer_settime',
11884                 'timerfd_gettime',
11885                 'timerfd_settime',
11886                 'unlink',
11887                 'unlinkat',
11888                 'uselib',
11889                 'utime',
11890                 'utimensat',
11891                 'vmsplice',
11892                 'write',
11893                 'writev']),
11894  'utimensat': set(['accept4',
11895                    'bind',
11896                    'bpf',
11897                    'clock_nanosleep',
11898                    'connect',
11899                    'copy_file_range',
11900                    'epoll_ctl',
11901                    'epoll_wait',
11902                    'faccessat',
11903                    'fallocate',
11904                    'fchdir',
11905                    'fchmod',
11906                    'fchmodat',
11907                    'fchown',
11908                    'fchownat',
11909                    'fcntl',
11910                    'fcntl64',
11911                    'fdatasync',
11912                    'fgetxattr',
11913                    'flistxattr',
11914                    'flock',
11915                    'fremovexattr',
11916                    'fsetxattr',
11917                    'fstat',
11918                    'fstatfs',
11919                    'fstatfs64',
11920                    'fsync',
11921                    'ftruncate',
11922                    'futex',
11923                    'futimesat',
11924                    'getdents',
11925                    'getdents64',
11926                    'getpeername',
11927                    'getsockname',
11928                    'getsockopt',
11929                    'inotify_add_watch',
11930                    'inotify_rm_watch',
11931                    'io_getevents',
11932                    'ioctl',
11933                    'linkat',
11934                    'listen',
11935                    'llseek',
11936                    'lseek',
11937                    'memfd_create',
11938                    'mq_getsetattr',
11939                    'mq_notify',
```



```
11940                    'mq_timedreceive',
11941                    'mq_timedsend',
11942                    'mq_unlink',
11943                    'nanosleep',
11944                    'newfstat',
11945                    'old_readdir',
11946                    'perf_event_open',
11947                    'poll',
11948                    'ppoll',
11949                    'pread64',
11950                    'preadv',
11951                    'preadv2',
11952                    'preadv64',
11953                    'preadv64v2',
11954                    'pselect6',
11955                    'pwrite64',
11956                    'pwritev',
11957                    'pwritev2',
11958                    'pwritev64',
11959                    'pwritev64v2',
11960                    'read',
11961                    'readahead',
11962                    'readlinkat',
11963                    'readv',
11964                    'recvfrom',
11965                    'recvmmsg',
11966                    'rt_sigtimedwait',
11967                    'sched_rr_get_interval',
11968                    'select',
11969                    'semtimedop',
11970                    'sendfile',
11971                    'sendfile64',
11972                    'sendto',
11973                    'setsockopt',
11974                    'settimeofday',
11975                    'shutdown',
11976                    'signalfd4',
11977                    'splice',
11978                    'stime',
11979                    'swapoff',
11980                    'swapon',
11981                    'sync_file_range',
11982                    'syncfs',
11983                    'tee',
11984                    'timer_gettime',
11985                    'timer_settime',
11986                    'timerfd_gettime',
11987                    'timerfd_settime',
11988                    'unlink',
11989                    'unlinkat',
11990                    'uselib',
11991                    'utime',
11992                    'utimensat',
11993                    'vmsplice',
11994                    'write',
11995                    'writev']),
11996  'vmsplice': set(['accept4',
11997                   'acct',
11998                   'bind',
11999                   'bpf',
12000                   'connect',
12001                   'copy_file_range',
12002                   'dup',
12003                   'dup3',
12004                   'epoll_create1',
12005                   'epoll_ctl',
```

```
                    'epoll_wait',
                    'eventfd2',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setns',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
```



```
                    'shutdown',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'write': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
```

```
12138                 'pwritev64v2',
12139                 'read',
12140                 'readahead',
12141                 'readv',
12142                 'recvfrom',
12143                 'sendfile',
12144                 'sendfile64',
12145                 'sendto',
12146                 'setsockopt',
12147                 'shutdown',
12148                 'signalfd4',
12149                 'splice',
12150                 'sync_file_range',
12151                 'syncfs',
12152                 'tee',
12153                 'utime',
12154                 'utimensat',
12155                 'vmsplice',
12156                 'write',
12157                 'writev']),
12158  'writev': set(['accept4',
12159                 'bind',
12160                 'bpf',
12161                 'connect',
12162                 'copy_file_range',
12163                 'epoll_ctl',
12164                 'epoll_wait',
12165                 'fallocate',
12166                 'fchdir',
12167                 'fchmod',
12168                 'fchown',
12169                 'fcntl',
12170                 'fcntl64',
12171                 'fdatasync',
12172                 'fgetxattr',
12173                 'flistxattr',
12174                 'flock',
12175                 'fremovexattr',
12176                 'fsetxattr',
12177                 'fstatfs',
12178                 'fstatfs64',
12179                 'fsync',
12180                 'ftruncate',
12181                 'futimesat',
12182                 'getdents',
12183                 'getdents64',
12184                 'getpeername',
12185                 'getsockname',
12186                 'getsockopt',
12187                 'inotify_add_watch',
12188                 'inotify_rm_watch',
12189                 'ioctl',
12190                 'listen',
12191                 'llseek',
12192                 'lseek',
12193                 'mq_getsetattr',
12194                 'mq_notify',
12195                 'mq_timedreceive',
12196                 'mq_timedsend',
12197                 'old_readdir',
12198                 'perf_event_open',
12199                 'pread64',
12200                 'preadv',
12201                 'preadv2',
12202                 'preadv64',
12203                 'preadv64v2',
```



```
12204                 'pwrite64',
12205                 'pwritev',
12206                 'pwritev2',
12207                 'pwritev64',
12208                 'pwritev64v2',
12209                 'read',
12210                 'readahead',
12211                 'readv',
12212                 'recvfrom',
12213                 'sendfile',
12214                 'sendfile64',
12215                 'sendto',
12216                 'setsockopt',
12217                 'shutdown',
12218                 'signalfd4',
12219                 'splice',
12220                 'sync_file_range',
12221                 'syncfs',
12222                 'tee',
12223                 'utime',
12224                 'utimensat',
12225                 'vmsplice',
12226                 'write',
12227                 'writev'])}
```

```
**********************************************************
  310366 Fri Dec 21 15:00:32 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/with_structs/i
mplicit_dependencies.pretty
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
{'acct': set(['accept4',
              'dup',
              'dup3',
              'epoll_create1',
              'epoll_ctl',
              'eventfd2',
              'flock',
              'memfd_create',
              'mmap_pgoff',
              'mq_open',
              'open',
              'open_by_handle_at',
              'openat',
              'perf_event_open',
              'pipe2',
              'remap_file_pages',
              'setns',
              'shmat',
              'shmctl',
              'shmdt',
              'socket',
              'socketpair',
              'swapoff',
              'swapon',
              'uselib']),
 'alarm': set(['adjtimex',
               'clock_adjtime',
               'getitimer',
               'getrusage',
               'ppoll',
               'select',
               'setitimer',
               'settimeofday',
               'wait4',
               'waitid']),
 'bpf': set(['accept4',
             'acct',
             'capget',
             'clone',
             'dup',
             'dup3',
             'epoll_create1',
             'epoll_ctl',
             'eventfd2',
             'flock',
             'fork',
             'get_robust_list',
             'getitimer',
             'getpgid',
             'getppid',
             'getpriority',
             'getrusage',
             'getsid',
             'ioperm',
             'ioprio_get',
             'ioprio_set',
             'keyctl',
             'kill',
             'memfd_create',
```



```
             'migrate_pages',
             'mmap_pgoff',
             'move_pages',
             'mq_open',
             'mq_timedreceive',
             'mq_timedsend',
             'msgrcv',
             'open',
             'open_by_handle_at',
             'openat',
             'perf_event_open',
             'pipe2',
             'prctl',
             'prlimit64',
             'ptrace',
             'remap_file_pages',
             'rt_sigaction',
             'rt_sigprocmask',
             'rt_sigtimedwait',
             'sched_getaffinity',
             'sched_getattr',
             'sched_getparam',
             'sched_getscheduler',
             'sched_rr_get_interval',
             'sched_setaffinity',
             'sched_setattr',
             'sched_setparam',
             'sched_setscheduler',
             'semtimedop',
             'setitimer',
             'setns',
             'setpgid',
             'setpriority',
             'setsid',
             'shmat',
             'shmctl',
             'shmdt',
             'sigaction',
             'sigaltstack',
             'signal',
             'socket',
             'socketpair',
             'swapoff',
             'swapon',
             'umount',
             'uselib',
             'vfork']),
 'brk': set(['get_mempolicy',
             'getrusage',
             'io_cancel',
             'io_destroy',
             'io_getevents',
             'io_setup',
             'madvise',
             'mbind',
             'migrate_pages',
             'mincore',
             'mlockall',
             'modify_ldt',
             'move_pages',
             'mprotect',
             'mremap',
             'munlock',
             'munlockall',
             'pkey_mprotect',
             'prctl',
```

```
126             'remap_file_pages',
127             'shmdt',
128             'swapoff']),
129  'capset': set(['capget',
130                 'clone',
131                 'fork',
132                 'get_robust_list',
133                 'getitimer',
134                 'getpgid',
135                 'getppid',
136                 'getpriority',
137                 'getrusage',
138                 'getsid',
139                 'ioprio_get',
140                 'ioprio_set',
141                 'keyctl',
142                 'kill',
143                 'migrate_pages',
144                 'move_pages',
145                 'mq_timedreceive',
146                 'mq_timedsend',
147                 'msgrcv',
148                 'perf_event_open',
149                 'prctl',
150                 'prlimit64',
151                 'ptrace',
152                 'rt_sigaction',
153                 'rt_sigprocmask',
154                 'rt_sigtimedwait',
155                 'sched_getaffinity',
156                 'sched_getattr',
157                 'sched_getparam',
158                 'sched_getscheduler',
159                 'sched_rr_get_interval',
160                 'sched_setaffinity',
161                 'sched_setattr',
162                 'sched_setparam',
163                 'sched_setscheduler',
164                 'semtimedop',
165                 'setitimer',
166                 'setns',
167                 'setpgid',
168                 'setpriority',
169                 'setsid',
170                 'sigaction',
171                 'sigaltstack',
172                 'signal',
173                 'umount',
174                 'vfork']),
175  'clock_adjtime': set(['clock_getres',
176                        'clock_gettime',
177                        'clock_nanosleep',
178                        'clock_settime',
179                        'timer_create',
180                        'timer_delete',
181                        'timer_gettime',
182                        'timer_settime']),
183  'clock_nanosleep': set(['clock_adjtime',
184                          'clock_getres',
185                          'clock_gettime',
186                          'clock_settime',
187                          'epoll_wait',
188                          'faccessat',
189                          'fchmod',
190                          'fchmodat',
191                          'fchown',
```



```
192                          'fchownat',
193                          'fstat',
194                          'ftruncate',
195                          'futex',
196                          'futimesat',
197                          'inotify_add_watch',
198                          'io_getevents',
199                          'ioctl',
200                          'linkat',
201                          'memfd_create',
202                          'mq_getsetattr',
203                          'mq_notify',
204                          'mq_timedreceive',
205                          'mq_timedsend',
206                          'mq_unlink',
207                          'nanosleep',
208                          'newfstat',
209                          'poll',
210                          'ppoll',
211                          'pselect6',
212                          'readlinkat',
213                          'recvmmsg',
214                          'rt_sigtimedwait',
215                          'sched_rr_get_interval',
216                          'select',
217                          'semtimedop',
218                          'sendfile',
219                          'sendfile64',
220                          'settimeofday',
221                          'stime',
222                          'swapoff',
223                          'swapon',
224                          'timer_create',
225                          'timer_delete',
226                          'timer_gettime',
227                          'timer_settime',
228                          'timerfd_gettime',
229                          'timerfd_settime',
230                          'unlink',
231                          'unlinkat',
232                          'uselib',
233                          'utime']),
234  'clock_settime': set(['clock_adjtime',
235                        'clock_getres',
236                        'clock_gettime',
237                        'clock_nanosleep',
238                        'timer_create',
239                        'timer_delete',
240                        'timer_gettime',
241                        'timer_settime']),
242  'copy_file_range': set(['accept4',
243                          'bind',
244                          'bpf',
245                          'connect',
246                          'epoll_ctl',
247                          'epoll_wait',
248                          'fallocate',
249                          'fchdir',
250                          'fchmod',
251                          'fchown',
252                          'fcntl',
253                          'fcntl64',
254                          'fdatasync',
255                          'fgetxattr',
256                          'flistxattr',
257                          'flock',
```

```
                        'fremovexattr',
                        'fsetxattr',
                        'fstatfs',
                        'fstatfs64',
                        'fsync',
                        'ftruncate',
                        'futimesat',
                        'getdents',
                        'getdents64',
                        'getpeername',
                        'getsockname',
                        'getsockopt',
                        'inotify_add_watch',
                        'inotify_rm_watch',
                        'ioctl',
                        'listen',
                        'llseek',
                        'lseek',
                        'mq_getsetattr',
                        'mq_notify',
                        'mq_timedreceive',
                        'mq_timedsend',
                        'old_readdir',
                        'perf_event_open',
                        'pread64',
                        'preadv',
                        'preadv2',
                        'preadv64',
                        'preadv64v2',
                        'pwrite64',
                        'pwritev',
                        'pwritev2',
                        'pwritev64',
                        'pwritev64v2',
                        'read',
                        'readahead',
                        'readv',
                        'recvfrom',
                        'sendfile',
                        'sendfile64',
                        'sendto',
                        'setsockopt',
                        'shutdown',
                        'signalfd4',
                        'splice',
                        'sync_file_range',
                        'syncfs',
                        'tee',
                        'utime',
                        'utimensat',
                        'vmsplice',
                        'write',
                        'writev']),
    'delete_module': set(['finit_module', 'init_module']),
    'dup3': set(['dup2', 'select', 'unshare']),
    'epoll_create1': set(['capget',
                          'capset',
                          'clone',
                          'epoll_ctl',
                          'epoll_wait',
                          'faccessat',
                          'fork',
                          'get_robust_list',
                          'getgroups',
                          'getgroups16',
                          'getitimer',
```



```
                          'getpgid',
                          'getppid',
                          'getpriority',
                          'getresgid',
                          'getresgid16',
                          'getresuid',
                          'getresuid16',
                          'getrusage',
                          'getsid',
                          'ioprio_get',
                          'ioprio_set',
                          'keyctl',
                          'kill',
                          'migrate_pages',
                          'move_pages',
                          'mq_timedreceive',
                          'mq_timedsend',
                          'msgrcv',
                          'perf_event_open',
                          'prctl',
                          'prlimit64',
                          'ptrace',
                          'rt_sigaction',
                          'rt_sigprocmask',
                          'rt_sigtimedwait',
                          'sched_getaffinity',
                          'sched_getattr',
                          'sched_getparam',
                          'sched_getscheduler',
                          'sched_rr_get_interval',
                          'sched_setaffinity',
                          'sched_setattr',
                          'sched_setparam',
                          'sched_setscheduler',
                          'semtimedop',
                          'setfsgid',
                          'setfsuid',
                          'setgid',
                          'setitimer',
                          'setns',
                          'setpgid',
                          'setpriority',
                          'setregid',
                          'setresgid',
                          'setresuid',
                          'setreuid',
                          'setsid',
                          'setuid',
                          'sigaction',
                          'sigaltstack',
                          'signal',
                          'umount',
                          'unshare',
                          'vfork']),
    'epoll_ctl': set(['accept4',
                      'acct',
                      'bind',
                      'bpf',
                      'brk',
                      'capget',
                      'clone',
                      'connect',
                      'copy_file_range',
                      'delete_module',
                      'dup',
                      'dup2',
```

```
390                     'dup3',
391                     'epoll_create1',
392                     'epoll_wait',
393                     'eventfd2',
394                     'exit_group',
395                     'faccessat',
396                     'fallocate',
397                     'fchdir',
398                     'fchmod',
399                     'fchmodat',
400                     'fchown',
401                     'fchownat',
402                     'fcntl',
403                     'fcntl64',
404                     'fdatasync',
405                     'fgetxattr',
406                     'finit_module',
407                     'flistxattr',
408                     'flock',
409                     'fork',
410                     'fremovexattr',
411                     'fsetxattr',
412                     'fstatfs',
413                     'fstatfs64',
414                     'fsync',
415                     'ftruncate',
416                     'futimesat',
417                     'get_curr_temp',
418                     'get_mempolicy',
419                     'get_robust_list',
420                     'get_trip_temp',
421                     'getcwd',
422                     'getdents',
423                     'getdents64',
424                     'getgroups',
425                     'getgroups16',
426                     'getitimer',
427                     'getpeername',
428                     'getpgid',
429                     'getppid',
430                     'getpriority',
431                     'getrusage',
432                     'getsid',
433                     'getsockname',
434                     'getsockopt',
435                     'init_module',
436                     'inotify_add_watch',
437                     'inotify_init1',
438                     'inotify_rm_watch',
439                     'io_cancel',
440                     'io_destroy',
441                     'io_getevents',
442                     'io_setup',
443                     'io_submit',
444                     'ioctl',
445                     'ioprio_get',
446                     'ioprio_set',
447                     'kexec_load',
448                     'keyctl',
449                     'kill',
450                     'linkat',
451                     'listen',
452                     'llseek',
453                     'lookup_dcookie',
454                     'lseek',
455                     'madvise',
```



```
456                     'mbind',
457                     'memfd_create',
458                     'migrate_pages',
459                     'mincore',
460                     'mkdirat',
461                     'mknodat',
462                     'mlockall',
463                     'mmap_pgoff',
464                     'modify_ldt',
465                     'move_pages',
466                     'mprotect',
467                     'mq_getsetattr',
468                     'mq_notify',
469                     'mq_open',
470                     'mq_timedreceive',
471                     'mq_timedsend',
472                     'mq_unlink',
473                     'mremap',
474                     'msgctl',
475                     'msgrcv',
476                     'msgsnd',
477                     'munlock',
478                     'munlockall',
479                     'old_readdir',
480                     'open',
481                     'open_by_handle_at',
482                     'openat',
483                     'perf_event_open',
484                     'pipe2',
485                     'pivot_root',
486                     'pkey_mprotect',
487                     'prctl',
488                     'pread64',
489                     'preadv',
490                     'preadv2',
491                     'preadv64',
492                     'preadv64v2',
493                     'prlimit64',
494                     'ptrace',
495                     'pwrite64',
496                     'pwritev',
497                     'pwritev2',
498                     'pwritev64',
499                     'pwritev64v2',
500                     'quotactl',
501                     'read',
502                     'readahead',
503                     'readlinkat',
504                     'readv',
505                     'reboot',
506                     'recvfrom',
507                     'remap_file_pages',
508                     'renameat2',
509                     'request_key',
510                     'rmdir',
511                     'rt_sigaction',
512                     'rt_sigprocmask',
513                     'rt_sigtimedwait',
514                     'sched_getaffinity',
515                     'sched_getattr',
516                     'sched_getparam',
517                     'sched_getscheduler',
518                     'sched_rr_get_interval',
519                     'sched_setaffinity',
520                     'sched_setattr',
521                     'sched_setparam',
```

```
                    'sched_setscheduler',
                    'sched_yield',
                    'semctl',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'set_trip_temp',
                    'setgid',
                    'setgroups',
                    'setgroups16',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setregid',
                    'setresgid',
                    'setresuid',
                    'setreuid',
                    'setsid',
                    'setsockopt',
                    'setuid',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'symlinkat',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'timer_create',
                    'timer_delete',
                    'timer_getoverrun',
                    'timer_gettime',
                    'timer_settime',
                    'timerfd_create',
                    'timerfd_gettime',
                    'timerfd_settime',
                    'umount',
                    'unlink',
                    'unlinkat',
                    'unshare',
                    'uselib',
                    'ustat',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
                    'write',
                    'writev']),
'epoll_wait': set(['accept4',
                   'acct',
                   'bind',
                   'bpf',
                   'capget',
                   'clone',
                   'connect',
```



```
                   'copy_file_range',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'fgetxattr',
                   'flistxattr',
                   'flock',
                   'fork',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'get_robust_list',
                   'getdents',
                   'getdents64',
                   'getitimer',
                   'getpeername',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'getsockname',
                   'getsockopt',
                   'inotify_add_watch',
                   'inotify_rm_watch',
                   'ioctl',
                   'ioperm',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'listen',
                   'llseek',
                   'lseek',
                   'memfd_create',
                   'migrate_pages',
                   'mmap_pgoff',
                   'move_pages',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_open',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'old_readdir',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'prctl',
                   'pread64',
                   'preadv',
```

```
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'prlimit64',
                    'ptrace',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'umount',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
                    'write',
                    'writev']),
 'faccessat': set(['accept4',
                   'acct',
                   'capget',
                   'clone',
                   'dup',
```



```
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'flock',
                   'fork',
                   'get_robust_list',
                   'getcwd',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'lookup_dcookie',
                   'memfd_create',
                   'migrate_pages',
                   'mmap_pgoff',
                   'move_pages',
                   'mq_open',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'pivot_root',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'quotactl',
                   'remap_file_pages',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setresuid',
                   'setreuid',
                   'setsid',
                   'setuid',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'socket',
```

```
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'umount',
                    'unshare',
                    'uselib',
                    'vfork']),
 'fallocate': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
```



```
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'fchdir': set(['accept4',
                'acct',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getcwd',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lookup_dcookie',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'pread64',
```

```
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'quotactl',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'unshare',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchmod': set(['accept4',
                'acct',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
```



```
                'ftruncate',
                'futimesat',
                'getcwd',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lookup_dcookie',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'quotactl',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'unshare',
                'uselib',
                'utime',
```

```
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchmodat': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'getcwd',
                  'lookup_dcookie',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pivot_root',
                  'quotactl',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'unshare',
                  'uselib']),
 'fchown': set(['accept4',
                'acct',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchmod',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getcwd',
                'getdents',
                'getdents64',
```



```
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lookup_dcookie',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'quotactl',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'unshare',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchownat': set(['accept4',
```

```
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'getcwd',
                  'lookup_dcookie',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pivot_root',
                  'quotactl',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'unshare',
                  'uselib']),
 'fcntl': set(['accept4',
               'acct',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'epoll_wait',
               'eventfd2',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
```



```
               'listen',
               'llseek',
               'lseek',
               'memfd_create',
               'mmap_pgoff',
               'mq_getsetattr',
               'mq_notify',
               'mq_open',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'open',
               'open_by_handle_at',
               'openat',
               'perf_event_open',
               'pipe2',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'remap_file_pages',
               'sendfile',
               'sendfile64',
               'sendto',
               'setns',
               'setsockopt',
               'shmat',
               'shmctl',
               'shmdt',
               'shutdown',
               'signalfd4',
               'socket',
               'socketpair',
               'splice',
               'swapoff',
               'swapon',
               'sync_file_range',
               'syncfs',
               'tee',
               'uselib',
               'utime',
               'utimensat',
               'vmsplice',
               'write',
               'writev']),
 'fcntl64': set(['accept4',
                 'acct',
                 'bind',
                 'bpf',
                 'connect',
                 'copy_file_range',
                 'dup',
                 'dup3',
                 'epoll_create1',
                 'epoll_ctl',
                 'epoll_wait',
```

```
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
```



```
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fgetxattr': set(['accept4',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'epoll_ctl',
                   'epoll_wait',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'flistxattr',
                   'flock',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'getdents',
                   'getdents64',
                   'getpeername',
                   'getsockname',
                   'getsockopt',
                   'inotify_add_watch',
                   'inotify_rm_watch',
                   'ioctl',
                   'listen',
                   'llseek',
                   'lseek',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'old_readdir',
                   'perf_event_open',
                   'pread64',
                   'preadv',
                   'preadv2',
                   'preadv64',
                   'preadv64v2',
                   'pwrite64',
                   'pwritev',
                   'pwritev2',
                   'pwritev64',
                   'pwritev64v2',
                   'read',
                   'readahead',
```

```
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'finit_module': set(['delete_module', 'init_module']),
 'flistxattr': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
```



```
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'flock': set(['accept4',
               'acct',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'epoll_wait',
               'eventfd2',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'memfd_create',
               'mmap_pgoff',
               'mq_getsetattr',
               'mq_notify',
               'mq_open',
               'mq_timedreceive',
```

```
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'quotactl',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'umount',
                'uselib',
                'ustat',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
  'fremovexattr': set(['accept4',
                       'bind',
                       'bpf',
                       'connect',
                       'copy_file_range',
                       'epoll_ctl',
                       'epoll_wait',
                       'fallocate',
                       'fchdir',
                       'fchmod',
                       'fchown',
                       'fcntl',
                       'fcntl64',
                       'fdatasync',
                       'fgetxattr',
                       'flistxattr',
                       'flock',
```



```
                       'fsetxattr',
                       'fstatfs',
                       'fstatfs64',
                       'fsync',
                       'ftruncate',
                       'futimesat',
                       'getdents',
                       'getdents64',
                       'getpeername',
                       'getsockname',
                       'getsockopt',
                       'inotify_add_watch',
                       'inotify_rm_watch',
                       'ioctl',
                       'listen',
                       'llseek',
                       'lseek',
                       'mq_getsetattr',
                       'mq_notify',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'old_readdir',
                       'perf_event_open',
                       'pread64',
                       'preadv',
                       'preadv2',
                       'preadv64',
                       'preadv64v2',
                       'pwrite64',
                       'pwritev',
                       'pwritev2',
                       'pwritev64',
                       'pwritev64v2',
                       'read',
                       'readahead',
                       'readv',
                       'recvfrom',
                       'sendfile',
                       'sendfile64',
                       'sendto',
                       'setsockopt',
                       'shutdown',
                       'signalfd4',
                       'splice',
                       'sync_file_range',
                       'syncfs',
                       'tee',
                       'utime',
                       'utimensat',
                       'vmsplice',
                       'write',
                       'writev']),
  'fsetxattr': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
```

```
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
  'fstat': set(['lstat', 'newfstat', 'stat']),
  'fstatfs': set(['fstatfs64', 'statfs', 'statfs64', 'ustat']),
  'fstatfs64': set(['fstatfs', 'statfs', 'statfs64', 'ustat']),
  'ftruncate': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'faccessat',
```



```
                    'fchmod',
                    'fchmodat',
                    'fchown',
                    'fchownat',
                    'flock',
                    'inotify_add_watch',
                    'ioctl',
                    'linkat',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'mq_unlink',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'readlinkat',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'unlink',
                    'unlinkat',
                    'uselib']),
  'futex': set(['clock_nanosleep',
                'epoll_wait',
                'faccessat',
                'fchmod',
                'fchmodat',
                'fchown',
                'fchownat',
                'fstat',
                'ftruncate',
                'futimesat',
                'inotify_add_watch',
                'io_getevents',
                'ioctl',
                'linkat',
                'memfd_create',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'nanosleep',
                'newfstat',
                'poll',
                'ppoll',
                'pselect6',
                'readlinkat',
                'recvmmsg',
                'rt_sigtimedwait',
                'sched_rr_get_interval',
                'select',
```

```
                'semtimedop',
                'sendfile',
                'sendfile64',
                'settimeofday',
                'stime',
                'swapoff',
                'swapon',
                'timer_gettime',
                'timer_settime',
                'timerfd_gettime',
                'timerfd_settime',
                'unlink',
                'unlinkat',
                'uselib',
                'utime']),
 'futimesat': set(['adjtimex',
                   'alarm',
                   'clock_adjtime',
                   'clock_nanosleep',
                   'epoll_wait',
                   'faccessat',
                   'fchmod',
                   'fchmodat',
                   'fchown',
                   'fchownat',
                   'fstat',
                   'ftruncate',
                   'futex',
                   'getitimer',
                   'getrusage',
                   'inotify_add_watch',
                   'io_getevents',
                   'ioctl',
                   'linkat',
                   'memfd_create',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'mq_unlink',
                   'nanosleep',
                   'newfstat',
                   'poll',
                   'ppoll',
                   'pselect6',
                   'readlinkat',
                   'recvmmsg',
                   'rt_sigtimedwait',
                   'sched_rr_get_interval',
                   'select',
                   'semtimedop',
                   'sendfile',
                   'sendfile64',
                   'setitimer',
                   'settimeofday',
                   'stime',
                   'swapoff',
                   'swapon',
                   'timer_gettime',
                   'timer_settime',
                   'timerfd_gettime',
                   'timerfd_settime',
                   'unlink',
                   'unlinkat',
                   'uselib',
                   'utime',
```



```
                   'wait4',
                   'waitid']),
 'get_mempolicy': set(['brk',
                       'capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'madvise',
                       'mbind',
                       'migrate_pages',
                       'mincore',
                       'mlockall',
                       'move_pages',
                       'mprotect',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'mremap',
                       'msgrcv',
                       'munlock',
                       'munlockall',
                       'perf_event_open',
                       'pkey_mprotect',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'remap_file_pages',
                       'rt_sigaction',
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getattr',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setattr',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'set_mempolicy',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'shmdt',
                       'sigaction',
                       'sigaltstack',
                       'signal',
                       'umount',
                       'vfork']),
 'getcwd': set(['accept4',
                'acct',
                'dup',
                'dup3',
                'epoll_create1',
```

```
                'epoll_ctl',
                'eventfd2',
                'flock',
                'ftruncate',
                'linkat',
                'lookup_dcookie',
                'memfd_create',
                'mkdirat',
                'mknodat',
                'mmap_pgoff',
                'mq_open',
                'mq_unlink',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'quotactl',
                'remap_file_pages',
                'renameat2',
                'rmdir',
                'setns',
                'shmat',
                'shmctl',
                'shmdt',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'symlinkat',
                'umount',
                'unlink',
                'unlinkat',
                'unshare',
                'uselib']),
 'getdents': set(['accept4',
                  'bind',
                  'bpf',
                  'capget',
                  'clone',
                  'connect',
                  'copy_file_range',
                  'epoll_ctl',
                  'epoll_wait',
                  'fallocate',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
                  'fork',
                  'fremovexattr',
                  'fsetxattr',
                  'fstatfs',
                  'fstatfs64',
                  'fsync',
                  'ftruncate',
                  'futimesat',
                  'get_robust_list',
                  'getdents64',
                  'getitimer',
```



```
                  'getpeername',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'inotify_rm_watch',
                  'ioctl',
                  'ioperm',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'listen',
                  'llseek',
                  'lseek',
                  'migrate_pages',
                  'move_pages',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'old_readdir',
                  'perf_event_open',
                  'prctl',
                  'pread64',
                  'preadv',
                  'preadv2',
                  'preadv64',
                  'preadv64v2',
                  'prlimit64',
                  'ptrace',
                  'pwrite64',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
```

```
                  'setsockopt',
                  'shutdown',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'signalfd4',
                  'splice',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'umount',
                  'utime',
                  'utimensat',
                  'vfork',
                  'vmsplice',
                  'write',
                  'writev']),
 'getdents64': set(['accept4',
                    'bind',
                    'bpf',
                    'capget',
                    'clone',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fork',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'get_robust_list',
                    'getdents',
                    'getitimer',
                    'getpeername',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'ioperm',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'listen',
                    'llseek',
                    'lseek',
```



```
                    'migrate_pages',
                    'move_pages',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'old_readdir',
                    'perf_event_open',
                    'prctl',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'prlimit64',
                    'ptrace',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'setsockopt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'umount',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
                    'write',
                    'writev']),
 'getegid': set(['capget',
                 'clone',
```

```
                  'fork',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'getegid16': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
```



```
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'geteuid': set(['capget',
                 'clone',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'migrate_pages',
                 'move_pages',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'msgrcv',
                 'perf_event_open',
                 'prctl',
                 'prlimit64',
                 'ptrace',
                 'rt_sigaction',
                 'rt_sigprocmask',
                 'rt_sigtimedwait',
                 'sched_getaffinity',
                 'sched_getattr',
                 'sched_getparam',
                 'sched_getscheduler',
                 'sched_rr_get_interval',
                 'sched_setaffinity',
                 'sched_setattr',
                 'sched_setparam',
                 'sched_setscheduler',
                 'semtimedop',
                 'setitimer',
                 'setns',
                 'setpgid',
                 'setpriority',
                 'setsid',
                 'sigaction',
```

```
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'geteuid16': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'getgid': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
```



```
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'getgid16': set(['capget',
                  'clone',
                  'fork',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
```

```
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'getgroups': set(['capget',
                   'capset',
                   'clone',
                   'epoll_create1',
                   'faccessat',
                   'fork',
                   'get_robust_list',
                   'getgroups16',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getresgid',
                   'getresgid16',
                   'getresuid',
                   'getresuid16',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setfsgid',
                   'setfsuid',
                   'setgid',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setregid',
                   'setresgid',
                   'setresuid',
                   'setreuid',
                   'setsid',
```



```
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'getgroups16': set(['capget',
                     'capset',
                     'clone',
                     'epoll_create1',
                     'faccessat',
                     'fork',
                     'get_robust_list',
                     'getgroups',
                     'getitimer',
                     'getpgid',
                     'getppid',
                     'getpriority',
                     'getresgid',
                     'getresgid16',
                     'getresuid',
                     'getresuid16',
                     'getrusage',
                     'getsid',
                     'ioprio_get',
                     'ioprio_set',
                     'keyctl',
                     'kill',
                     'migrate_pages',
                     'move_pages',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'msgrcv',
                     'perf_event_open',
                     'prctl',
                     'prlimit64',
                     'ptrace',
                     'rt_sigaction',
                     'rt_sigprocmask',
                     'rt_sigtimedwait',
                     'sched_getaffinity',
                     'sched_getattr',
                     'sched_getparam',
                     'sched_getscheduler',
                     'sched_rr_get_interval',
                     'sched_setaffinity',
                     'sched_setattr',
                     'sched_setparam',
                     'sched_setscheduler',
                     'semtimedop',
                     'setfsgid',
                     'setfsuid',
                     'setgid',
                     'setgroups',
                     'setgroups16',
                     'setitimer',
                     'setns',
                     'setpgid',
                     'setpriority',
                     'setregid',
                     'setresgid',
                     'setresuid',
                     'setreuid',
                     'setsid',
                     'setuid',
```

```
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'unshare',
                    'vfork']),
 'getitimer': set(['exit_group', 'setitimer', 'timer_create']),
 'getppid': set(['capget',
                 'clone',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'migrate_pages',
                 'move_pages',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'msgrcv',
                 'perf_event_open',
                 'prctl',
                 'prlimit64',
                 'ptrace',
                 'rt_sigaction',
                 'rt_sigprocmask',
                 'rt_sigtimedwait',
                 'sched_getaffinity',
                 'sched_getattr',
                 'sched_getparam',
                 'sched_getscheduler',
                 'sched_rr_get_interval',
                 'sched_setaffinity',
                 'sched_setattr',
                 'sched_setparam',
                 'sched_setscheduler',
                 'semtimedop',
                 'setitimer',
                 'setns',
                 'setpgid',
                 'setpriority',
                 'setsid',
                 'sigaction',
                 'sigaltstack',
                 'signal',
                 'umount',
                 'vfork']),
 'getpriority': set(['capget',
                     'clone',
                     'fork',
                     'get_robust_list',
                     'getitimer',
                     'getpgid',
                     'getppid',
                     'getrusage',
                     'getsid',
                     'ioprio_get',
                     'ioprio_set',
                     'keyctl',
                     'kill',
                     'migrate_pages',
```



```
                     'move_pages',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'msgrcv',
                     'perf_event_open',
                     'prctl',
                     'prlimit64',
                     'ptrace',
                     'rt_sigaction',
                     'rt_sigprocmask',
                     'rt_sigtimedwait',
                     'sched_getaffinity',
                     'sched_getattr',
                     'sched_getparam',
                     'sched_getscheduler',
                     'sched_rr_get_interval',
                     'sched_setaffinity',
                     'sched_setattr',
                     'sched_setparam',
                     'sched_setscheduler',
                     'semtimedop',
                     'setitimer',
                     'setns',
                     'setpgid',
                     'setpriority',
                     'setresuid',
                     'setreuid',
                     'setsid',
                     'setuid',
                     'sigaction',
                     'sigaltstack',
                     'signal',
                     'umount',
                     'vfork']),
 'getresgid': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
```

```
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'getresgid16': set(['capget',
                     'clone',
                     'fork',
                     'get_robust_list',
                     'getitimer',
                     'getpgid',
                     'getppid',
                     'getpriority',
                     'getrusage',
                     'getsid',
                     'ioprio_get',
                     'ioprio_set',
                     'keyctl',
                     'kill',
                     'migrate_pages',
                     'move_pages',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'msgrcv',
                     'perf_event_open',
                     'prctl',
                     'prlimit64',
                     'ptrace',
                     'rt_sigaction',
                     'rt_sigprocmask',
                     'rt_sigtimedwait',
                     'sched_getaffinity',
                     'sched_getattr',
                     'sched_getparam',
                     'sched_getscheduler',
                     'sched_rr_get_interval',
                     'sched_setaffinity',
                     'sched_setattr',
                     'sched_setparam',
                     'sched_setscheduler',
                     'semtimedop',
                     'setitimer',
                     'setns',
                     'setpgid',
                     'setpriority',
                     'setsid',
                     'sigaction',
                     'sigaltstack',
                     'signal',
                     'umount',
                     'vfork']),
 'getresuid': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
```



```
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'getresuid16': set(['capget',
                     'clone',
                     'fork',
                     'get_robust_list',
                     'getitimer',
                     'getpgid',
                     'getppid',
                     'getpriority',
                     'getrusage',
                     'getsid',
                     'ioprio_get',
                     'ioprio_set',
                     'keyctl',
                     'kill',
                     'migrate_pages',
                     'move_pages',
                     'mq_timedreceive',
                     'mq_timedsend',
                     'msgrcv',
                     'perf_event_open',
                     'prctl',
                     'prlimit64',
                     'ptrace',
                     'rt_sigaction',
                     'rt_sigprocmask',
                     'rt_sigtimedwait',
```

```
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'getrlimit': set(['old_getrlimit', 'prlimit64', 'setrlimit']),
 'getrusage': set(['exit_group', 'timer_create']),
 'getsockopt': set(['accept4']),
 'getuid': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
```



```
                'signal',
                'umount',
                'vfork']),
 'getuid16': set(['capget',
                  'clone',
                  'fork',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'getxattr': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'getcwd',
                  'lookup_dcookie',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
```

```
                  'pipe2',
                  'pivot_root',
                  'quotactl',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'unshare',
                  'uselib']),
 'init_module': set(['delete_module', 'finit_module']),
 'inotify_add_watch': set(['accept4',
                           'acct',
                           'bind',
                           'bpf',
                           'connect',
                           'copy_file_range',
                           'dup',
                           'dup3',
                           'epoll_create1',
                           'epoll_ctl',
                           'epoll_wait',
                           'eventfd2',
                           'fallocate',
                           'fchdir',
                           'fchmod',
                           'fchown',
                           'fcntl',
                           'fcntl64',
                           'fdatasync',
                           'fgetxattr',
                           'flistxattr',
                           'flock',
                           'fremovexattr',
                           'fsetxattr',
                           'fstatfs',
                           'fstatfs64',
                           'fsync',
                           'ftruncate',
                           'futimesat',
                           'getcwd',
                           'getdents',
                           'getdents64',
                           'getpeername',
                           'getsockname',
                           'getsockopt',
                           'inotify_rm_watch',
                           'ioctl',
                           'linkat',
                           'listen',
                           'llseek',
                           'lseek',
                           'memfd_create',
                           'mkdirat',
                           'mknodat',
                           'mmap_pgoff',
                           'mq_getsetattr',
                           'mq_notify',
                           'mq_open',
                           'mq_timedreceive',
                           'mq_timedsend',
                           'mq_unlink',
```



```
                           'old_readdir',
                           'open',
                           'open_by_handle_at',
                           'openat',
                           'perf_event_open',
                           'pipe2',
                           'pivot_root',
                           'pread64',
                           'preadv',
                           'preadv2',
                           'preadv64',
                           'preadv64v2',
                           'pwrite64',
                           'pwritev',
                           'pwritev2',
                           'pwritev64',
                           'pwritev64v2',
                           'read',
                           'readahead',
                           'readv',
                           'recvfrom',
                           'remap_file_pages',
                           'renameat2',
                           'rmdir',
                           'sendfile',
                           'sendfile64',
                           'sendto',
                           'setns',
                           'setsockopt',
                           'shmat',
                           'shmctl',
                           'shmdt',
                           'shutdown',
                           'signalfd4',
                           'socket',
                           'socketpair',
                           'splice',
                           'swapoff',
                           'swapon',
                           'symlinkat',
                           'sync_file_range',
                           'syncfs',
                           'tee',
                           'unlink',
                           'unlinkat',
                           'uselib',
                           'utime',
                           'utimensat',
                           'vmsplice',
                           'write',
                           'writev']),
 'inotify_init1': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'inotify_add_watch',
                       'inotify_rm_watch',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
```

```
3162                         'kill',
3163                         'migrate_pages',
3164                         'move_pages',
3165                         'mq_timedreceive',
3166                         'mq_timedsend',
3167                         'msgrcv',
3168                         'perf_event_open',
3169                         'prctl',
3170                         'prlimit64',
3171                         'ptrace',
3172                         'rt_sigaction',
3173                         'rt_sigprocmask',
3174                         'rt_sigtimedwait',
3175                         'sched_getaffinity',
3176                         'sched_getattr',
3177                         'sched_getparam',
3178                         'sched_getscheduler',
3179                         'sched_rr_get_interval',
3180                         'sched_setaffinity',
3181                         'sched_setattr',
3182                         'sched_setparam',
3183                         'sched_setscheduler',
3184                         'semtimedop',
3185                         'setitimer',
3186                         'setns',
3187                         'setpgid',
3188                         'setpriority',
3189                         'setsid',
3190                         'sigaction',
3191                         'sigaltstack',
3192                         'signal',
3193                         'umount',
3194                         'vfork']),
3195  'inotify_rm_watch': set(['accept4',
3196                           'acct',
3197                           'bind',
3198                           'bpf',
3199                           'connect',
3200                           'copy_file_range',
3201                           'dup',
3202                           'dup3',
3203                           'epoll_create1',
3204                           'epoll_ctl',
3205                           'epoll_wait',
3206                           'eventfd2',
3207                           'fallocate',
3208                           'fchdir',
3209                           'fchmod',
3210                           'fchown',
3211                           'fcntl',
3212                           'fcntl64',
3213                           'fdatasync',
3214                           'fgetxattr',
3215                           'flistxattr',
3216                           'flock',
3217                           'fremovexattr',
3218                           'fsetxattr',
3219                           'fstatfs',
3220                           'fstatfs64',
3221                           'fsync',
3222                           'ftruncate',
3223                           'futimesat',
3224                           'getdents',
3225                           'getdents64',
3226                           'getpeername',
3227                           'getsockname',
```



```
3228                           'getsockopt',
3229                           'inotify_add_watch',
3230                           'ioctl',
3231                           'listen',
3232                           'llseek',
3233                           'lseek',
3234                           'memfd_create',
3235                           'mmap_pgoff',
3236                           'mq_getsetattr',
3237                           'mq_notify',
3238                           'mq_open',
3239                           'mq_timedreceive',
3240                           'mq_timedsend',
3241                           'old_readdir',
3242                           'open',
3243                           'open_by_handle_at',
3244                           'openat',
3245                           'perf_event_open',
3246                           'pipe2',
3247                           'pread64',
3248                           'preadv',
3249                           'preadv2',
3250                           'preadv64',
3251                           'preadv64v2',
3252                           'pwrite64',
3253                           'pwritev',
3254                           'pwritev2',
3255                           'pwritev64',
3256                           'pwritev64v2',
3257                           'read',
3258                           'readahead',
3259                           'readv',
3260                           'recvfrom',
3261                           'remap_file_pages',
3262                           'sendfile',
3263                           'sendfile64',
3264                           'sendto',
3265                           'setns',
3266                           'setsockopt',
3267                           'shmat',
3268                           'shmctl',
3269                           'shmdt',
3270                           'shutdown',
3271                           'signalfd4',
3272                           'socket',
3273                           'socketpair',
3274                           'splice',
3275                           'swapoff',
3276                           'swapon',
3277                           'sync_file_range',
3278                           'syncfs',
3279                           'tee',
3280                           'uselib',
3281                           'utime',
3282                           'utimensat',
3283                           'vmsplice',
3284                           'write',
3285                           'writev']),
3286  'io_cancel': set(['brk',
3287                    'get_mempolicy',
3288                    'getrusage',
3289                    'io_destroy',
3290                    'io_getevents',
3291                    'io_setup',
3292                    'io_submit',
3293                    'mbind',
```

```
                   'migrate_pages',
                   'mincore',
                   'modify_ldt',
                   'move_pages',
                   'mremap',
                   'prctl',
                   'remap_file_pages',
                   'shmdt',
                   'swapoff']),
 'io_destroy': set(['brk',
                    'get_mempolicy',
                    'getrusage',
                    'io_cancel',
                    'io_getevents',
                    'io_setup',
                    'io_submit',
                    'mbind',
                    'migrate_pages',
                    'mincore',
                    'modify_ldt',
                    'move_pages',
                    'mremap',
                    'prctl',
                    'remap_file_pages',
                    'shmdt',
                    'swapoff']),
 'io_getevents': set(['brk',
                      'capget',
                      'clone',
                      'fork',
                      'get_mempolicy',
                      'get_robust_list',
                      'getitimer',
                      'getpgid',
                      'getppid',
                      'getpriority',
                      'getrusage',
                      'getsid',
                      'io_cancel',
                      'io_destroy',
                      'io_setup',
                      'io_submit',
                      'ioprio_get',
                      'ioprio_set',
                      'keyctl',
                      'kill',
                      'mbind',
                      'migrate_pages',
                      'mincore',
                      'modify_ldt',
                      'move_pages',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'mremap',
                      'msgrcv',
                      'perf_event_open',
                      'prctl',
                      'prlimit64',
                      'ptrace',
                      'remap_file_pages',
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
```



```
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setsid',
                      'shmdt',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'swapoff',
                      'umount',
                      'vfork']),
 'io_setup': set(['io_cancel', 'io_destroy', 'io_getevents', 'io_submit']),
 'io_submit': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioperm',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
```

```
                  'vfork']),
 'ioctl': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'listen',
               'llseek',
               'lseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'sendfile',
               'sendfile64',
               'sendto',
               'setsockopt',
               'shutdown',
               'signalfd4',
               'splice',
               'sync_file_range',
               'syncfs',
               'tee',
               'utime',
```



```
               'utimensat',
               'vmsplice',
               'write',
               'writev']),
 'ioperm': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'ioprio_get': set(['capget',
                    'clone',
                    'fork',
                    'get_robust_list',
                    'getitimer',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
```

```
3558                    'mq_timedsend',
3559                    'msgrcv',
3560                    'perf_event_open',
3561                    'prctl',
3562                    'prlimit64',
3563                    'ptrace',
3564                    'rt_sigaction',
3565                    'rt_sigprocmask',
3566                    'rt_sigtimedwait',
3567                    'sched_getaffinity',
3568                    'sched_getattr',
3569                    'sched_getparam',
3570                    'sched_getscheduler',
3571                    'sched_rr_get_interval',
3572                    'sched_setaffinity',
3573                    'sched_setattr',
3574                    'sched_setparam',
3575                    'sched_setscheduler',
3576                    'semtimedop',
3577                    'setitimer',
3578                    'setns',
3579                    'setpgid',
3580                    'setpriority',
3581                    'setsid',
3582                    'sigaction',
3583                    'sigaltstack',
3584                    'signal',
3585                    'umount',
3586                    'vfork']),
3587  'ioprio_set': set(['capget',
3588                     'clone',
3589                     'fork',
3590                     'get_robust_list',
3591                     'getitimer',
3592                     'getpgid',
3593                     'getppid',
3594                     'getpriority',
3595                     'getrusage',
3596                     'getsid',
3597                     'ioprio_get',
3598                     'keyctl',
3599                     'kill',
3600                     'migrate_pages',
3601                     'move_pages',
3602                     'mq_timedreceive',
3603                     'mq_timedsend',
3604                     'msgrcv',
3605                     'perf_event_open',
3606                     'prctl',
3607                     'prlimit64',
3608                     'ptrace',
3609                     'rt_sigaction',
3610                     'rt_sigprocmask',
3611                     'rt_sigtimedwait',
3612                     'sched_getaffinity',
3613                     'sched_getattr',
3614                     'sched_getparam',
3615                     'sched_getscheduler',
3616                     'sched_rr_get_interval',
3617                     'sched_setaffinity',
3618                     'sched_setattr',
3619                     'sched_setparam',
3620                     'sched_setscheduler',
3621                     'semtimedop',
3622                     'setitimer',
3623                     'setns',
```



```
3624                     'setpgid',
3625                     'setpriority',
3626                     'setsid',
3627                     'sigaction',
3628                     'sigaltstack',
3629                     'signal',
3630                     'umount',
3631                     'vfork']),
3632  'keyctl': set(['capget',
3633                 'capset',
3634                 'clone',
3635                 'epoll_create1',
3636                 'faccessat',
3637                 'fork',
3638                 'get_robust_list',
3639                 'getgroups',
3640                 'getgroups16',
3641                 'getitimer',
3642                 'getpgid',
3643                 'getppid',
3644                 'getpriority',
3645                 'getresgid',
3646                 'getresgid16',
3647                 'getresuid',
3648                 'getresuid16',
3649                 'getrusage',
3650                 'getsid',
3651                 'ioprio_get',
3652                 'ioprio_set',
3653                 'kill',
3654                 'migrate_pages',
3655                 'move_pages',
3656                 'mq_timedreceive',
3657                 'mq_timedsend',
3658                 'msgrcv',
3659                 'perf_event_open',
3660                 'prctl',
3661                 'prlimit64',
3662                 'ptrace',
3663                 'request_key',
3664                 'rt_sigaction',
3665                 'rt_sigprocmask',
3666                 'rt_sigtimedwait',
3667                 'sched_getaffinity',
3668                 'sched_getattr',
3669                 'sched_getparam',
3670                 'sched_getscheduler',
3671                 'sched_rr_get_interval',
3672                 'sched_setaffinity',
3673                 'sched_setattr',
3674                 'sched_setparam',
3675                 'sched_setscheduler',
3676                 'semtimedop',
3677                 'setfsgid',
3678                 'setfsuid',
3679                 'setgid',
3680                 'setitimer',
3681                 'setns',
3682                 'setpgid',
3683                 'setpriority',
3684                 'setregid',
3685                 'setresgid',
3686                 'setresuid',
3687                 'setreuid',
3688                 'setsid',
3689                 'setuid',
```

```
3690                 'sigaction',
3691                 'sigaltstack',
3692                 'signal',
3693                 'umount',
3694                 'unshare',
3695                 'vfork']),
3696  'kill': set(['capget',
3697               'clone',
3698               'fork',
3699               'get_robust_list',
3700               'getitimer',
3701               'getpgid',
3702               'getppid',
3703               'getpriority',
3704               'getrusage',
3705               'getsid',
3706               'ioprio_get',
3707               'ioprio_set',
3708               'keyctl',
3709               'migrate_pages',
3710               'move_pages',
3711               'mq_timedreceive',
3712               'mq_timedsend',
3713               'msgrcv',
3714               'perf_event_open',
3715               'prctl',
3716               'prlimit64',
3717               'ptrace',
3718               'rt_sigaction',
3719               'rt_sigprocmask',
3720               'rt_sigtimedwait',
3721               'sched_getaffinity',
3722               'sched_getattr',
3723               'sched_getparam',
3724               'sched_getscheduler',
3725               'sched_rr_get_interval',
3726               'sched_setaffinity',
3727               'sched_setattr',
3728               'sched_setparam',
3729               'sched_setscheduler',
3730               'semtimedop',
3731               'setitimer',
3732               'setns',
3733               'setpgid',
3734               'setpriority',
3735               'setsid',
3736               'sigaction',
3737               'sigaltstack',
3738               'signal',
3739               'umount',
3740               'vfork']),
3741  'lgetxattr': set(['accept4',
3742                    'acct',
3743                    'dup',
3744                    'dup3',
3745                    'epoll_create1',
3746                    'epoll_ctl',
3747                    'eventfd2',
3748                    'flock',
3749                    'getcwd',
3750                    'lookup_dcookie',
3751                    'memfd_create',
3752                    'mmap_pgoff',
3753                    'mq_open',
3754                    'open',
3755                    'open_by_handle_at',
```



```
3756                    'openat',
3757                    'perf_event_open',
3758                    'pipe2',
3759                    'pivot_root',
3760                    'quotactl',
3761                    'remap_file_pages',
3762                    'setns',
3763                    'shmat',
3764                    'shmctl',
3765                    'shmdt',
3766                    'socket',
3767                    'socketpair',
3768                    'swapoff',
3769                    'swapon',
3770                    'unshare',
3771                    'uselib']),
3772  'linkat': set(['accept4',
3773                 'acct',
3774                 'dup',
3775                 'dup3',
3776                 'epoll_create1',
3777                 'epoll_ctl',
3778                 'eventfd2',
3779                 'flock',
3780                 'getcwd',
3781                 'lookup_dcookie',
3782                 'memfd_create',
3783                 'mmap_pgoff',
3784                 'mq_open',
3785                 'open',
3786                 'open_by_handle_at',
3787                 'openat',
3788                 'perf_event_open',
3789                 'pipe2',
3790                 'pivot_root',
3791                 'quotactl',
3792                 'remap_file_pages',
3793                 'setns',
3794                 'shmat',
3795                 'shmctl',
3796                 'shmdt',
3797                 'socket',
3798                 'socketpair',
3799                 'swapoff',
3800                 'swapon',
3801                 'unshare',
3802                 'uselib']),
3803  'listxattr': set(['accept4',
3804                    'acct',
3805                    'dup',
3806                    'dup3',
3807                    'epoll_create1',
3808                    'epoll_ctl',
3809                    'eventfd2',
3810                    'flock',
3811                    'getcwd',
3812                    'lookup_dcookie',
3813                    'memfd_create',
3814                    'mmap_pgoff',
3815                    'mq_open',
3816                    'open',
3817                    'open_by_handle_at',
3818                    'openat',
3819                    'perf_event_open',
3820                    'pipe2',
3821                    'pivot_root',
```

```
                   'quotactl',
                   'remap_file_pages',
                   'setns',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'socket',
                   'socketpair',
                   'swapoff',
                   'swapon',
                   'unshare',
                   'uselib']),
 'llistxattr': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'getcwd',
                    'lookup_dcookie',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_open',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pivot_root',
                    'quotactl',
                    'remap_file_pages',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'unshare',
                    'uselib']),
 'llseek': set(['accept4',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'epoll_ctl',
                'epoll_wait',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
```



```
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'lseek',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'perf_event_open',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'syncfs',
                'tee',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'lremovexattr': set(['accept4',
                      'acct',
                      'dup',
                      'dup3',
                      'epoll_create1',
                      'epoll_ctl',
                      'eventfd2',
                      'flock',
                      'getcwd',
                      'lookup_dcookie',
                      'memfd_create',
                      'mmap_pgoff',
                      'mq_open',
                      'open',
                      'open_by_handle_at',
                      'openat',
                      'perf_event_open',
                      'pipe2',
                      'pivot_root',
                      'quotactl',
```

```
                     'remap_file_pages',
                     'setns',
                     'shmat',
                     'shmctl',
                     'shmdt',
                     'socket',
                     'socketpair',
                     'swapoff',
                     'swapon',
                     'unshare',
                     'uselib']),
 'lseek': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'sendfile',
```



```
               'sendfile64',
               'sendto',
               'setsockopt',
               'shutdown',
               'signalfd4',
               'splice',
               'sync_file_range',
               'syncfs',
               'tee',
               'utime',
               'utimensat',
               'vmsplice',
               'write',
               'writev']),
 'lsetxattr': set(['accept4',
                   'acct',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'flock',
                   'getcwd',
                   'lookup_dcookie',
                   'memfd_create',
                   'mmap_pgoff',
                   'mq_open',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'pivot_root',
                   'quotactl',
                   'remap_file_pages',
                   'setns',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'socket',
                   'socketpair',
                   'swapoff',
                   'swapon',
                   'unshare',
                   'uselib']),
 'lstat': set(['fstat', 'newfstat', 'stat']),
 'madvise': set(['brk',
                 'get_mempolicy',
                 'mincore',
                 'mlockall',
                 'mprotect',
                 'mremap',
                 'munlock',
                 'munlockall',
                 'pkey_mprotect',
                 'prctl',
                 'remap_file_pages',
                 'shmdt']),
 'migrate_pages': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
```

```
                      'getrusage',
                      'getsid',
                      'ioperm',
                      'ioprio_get',
                      'ioprio_set',
                      'keyctl',
                      'kill',
                      'move_pages',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'msgrcv',
                      'perf_event_open',
                      'prctl',
                      'prlimit64',
                      'ptrace',
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setresuid',
                      'setreuid',
                      'setsid',
                      'setuid',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'umount',
                      'vfork']),
 'mincore': set(['brk',
                 'capget',
                 'clone',
                 'fork',
                 'get_mempolicy',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioperm',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'madvise',
                 'migrate_pages',
                 'mlockall',
                 'move_pages',
                 'mprotect',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'mremap',
```



```
                 'msgrcv',
                 'munlock',
                 'munlockall',
                 'perf_event_open',
                 'pkey_mprotect',
                 'prctl',
                 'prlimit64',
                 'ptrace',
                 'remap_file_pages',
                 'rt_sigaction',
                 'rt_sigprocmask',
                 'rt_sigtimedwait',
                 'sched_getaffinity',
                 'sched_getattr',
                 'sched_getparam',
                 'sched_getscheduler',
                 'sched_rr_get_interval',
                 'sched_setaffinity',
                 'sched_setattr',
                 'sched_setparam',
                 'sched_setscheduler',
                 'semtimedop',
                 'setitimer',
                 'setns',
                 'setpgid',
                 'setpriority',
                 'setsid',
                 'shmdt',
                 'sigaction',
                 'sigaltstack',
                 'signal',
                 'umount',
                 'vfork']),
 'mkdirat': set(['quotactl', 'swapon', 'syncfs', 'umount', 'ustat']),
 'mknodat': set(['quotactl', 'swapon', 'syncfs', 'umount', 'ustat']),
 'mlock': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'migrate_pages',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'msgrcv',
               'perf_event_open',
               'prctl',
               'prlimit64',
               'ptrace',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
```

```
4218                'sched_setaffinity',
4219                'sched_setattr',
4220                'sched_setparam',
4221                'sched_setscheduler',
4222                'semtimedop',
4223                'setitimer',
4224                'setns',
4225                'setpgid',
4226                'setpriority',
4227                'setsid',
4228                'sigaction',
4229                'sigaltstack',
4230                'signal',
4231                'umount',
4232                'vfork']),
4233  'mlock2': set(['capget',
4234                 'clone',
4235                 'fork',
4236                 'get_robust_list',
4237                 'getitimer',
4238                 'getpgid',
4239                 'getppid',
4240                 'getpriority',
4241                 'getrusage',
4242                 'getsid',
4243                 'ioprio_get',
4244                 'ioprio_set',
4245                 'keyctl',
4246                 'kill',
4247                 'migrate_pages',
4248                 'move_pages',
4249                 'mq_timedreceive',
4250                 'mq_timedsend',
4251                 'msgrcv',
4252                 'perf_event_open',
4253                 'prctl',
4254                 'prlimit64',
4255                 'ptrace',
4256                 'rt_sigaction',
4257                 'rt_sigprocmask',
4258                 'rt_sigtimedwait',
4259                 'sched_getaffinity',
4260                 'sched_getattr',
4261                 'sched_getparam',
4262                 'sched_getscheduler',
4263                 'sched_rr_get_interval',
4264                 'sched_setaffinity',
4265                 'sched_setattr',
4266                 'sched_setparam',
4267                 'sched_setscheduler',
4268                 'semtimedop',
4269                 'setitimer',
4270                 'setns',
4271                 'setpgid',
4272                 'setpriority',
4273                 'setsid',
4274                 'sigaction',
4275                 'sigaltstack',
4276                 'signal',
4277                 'umount',
4278                 'vfork']),
4279  'mlockall': set(['brk',
4280                   'capget',
4281                   'clone',
4282                   'fork',
4283                   'get_mempolicy',
```



```
4284                   'get_robust_list',
4285                   'getitimer',
4286                   'getpgid',
4287                   'getppid',
4288                   'getpriority',
4289                   'getrusage',
4290                   'getsid',
4291                   'io_cancel',
4292                   'io_destroy',
4293                   'io_getevents',
4294                   'io_setup',
4295                   'ioprio_get',
4296                   'ioprio_set',
4297                   'keyctl',
4298                   'kill',
4299                   'madvise',
4300                   'mbind',
4301                   'migrate_pages',
4302                   'mincore',
4303                   'modify_ldt',
4304                   'move_pages',
4305                   'mprotect',
4306                   'mq_timedreceive',
4307                   'mq_timedsend',
4308                   'mremap',
4309                   'msgrcv',
4310                   'munlock',
4311                   'munlockall',
4312                   'perf_event_open',
4313                   'personality',
4314                   'pkey_mprotect',
4315                   'prctl',
4316                   'prlimit64',
4317                   'ptrace',
4318                   'remap_file_pages',
4319                   'rt_sigaction',
4320                   'rt_sigprocmask',
4321                   'rt_sigtimedwait',
4322                   'sched_getaffinity',
4323                   'sched_getattr',
4324                   'sched_getparam',
4325                   'sched_getscheduler',
4326                   'sched_rr_get_interval',
4327                   'sched_setaffinity',
4328                   'sched_setattr',
4329                   'sched_setparam',
4330                   'sched_setscheduler',
4331                   'semtimedop',
4332                   'setitimer',
4333                   'setns',
4334                   'setpgid',
4335                   'setpriority',
4336                   'setsid',
4337                   'shmdt',
4338                   'sigaction',
4339                   'sigaltstack',
4340                   'signal',
4341                   'swapoff',
4342                   'umount',
4343                   'vfork']),
4344  'mmap_pgoff': set(['accept4',
4345                     'acct',
4346                     'dup',
4347                     'dup3',
4348                     'epoll_create1',
4349                     'epoll_ctl',
```

```
4350                    'eventfd2',
4351                    'flock',
4352                    'memfd_create',
4353                    'mq_open',
4354                    'open',
4355                    'open_by_handle_at',
4356                    'openat',
4357                    'perf_event_open',
4358                    'pipe2',
4359                    'remap_file_pages',
4360                    'setns',
4361                    'shmat',
4362                    'shmctl',
4363                    'shmdt',
4364                    'socket',
4365                    'socketpair',
4366                    'swapoff',
4367                    'swapon',
4368                    'uselib']),
4369  'modify_ldt': set(['brk',
4370                    'get_mempolicy',
4371                    'get_thread_area',
4372                    'getrusage',
4373                    'io_cancel',
4374                    'io_destroy',
4375                    'io_getevents',
4376                    'io_setup',
4377                    'mbind',
4378                    'migrate_pages',
4379                    'mincore',
4380                    'move_pages',
4381                    'mremap',
4382                    'prctl',
4383                    'remap_file_pages',
4384                    'set_thread_area',
4385                    'shmdt',
4386                    'swapoff']),
4387  'mount': set(['capget',
4388                'clone',
4389                'fork',
4390                'get_robust_list',
4391                'getitimer',
4392                'getpgid',
4393                'getppid',
4394                'getpriority',
4395                'getrusage',
4396                'getsid',
4397                'ioprio_get',
4398                'ioprio_set',
4399                'keyctl',
4400                'kill',
4401                'migrate_pages',
4402                'move_pages',
4403                'mq_timedreceive',
4404                'mq_timedsend',
4405                'msgrcv',
4406                'perf_event_open',
4407                'personality',
4408                'prctl',
4409                'prlimit64',
4410                'ptrace',
4411                'rt_sigaction',
4412                'rt_sigprocmask',
4413                'rt_sigtimedwait',
4414                'sched_getaffinity',
4415                'sched_getattr',
```



```
4416                'sched_getparam',
4417                'sched_getscheduler',
4418                'sched_rr_get_interval',
4419                'sched_setaffinity',
4420                'sched_setattr',
4421                'sched_setparam',
4422                'sched_setscheduler',
4423                'semtimedop',
4424                'setitimer',
4425                'setns',
4426                'setpgid',
4427                'setpriority',
4428                'setsid',
4429                'sigaction',
4430                'sigaltstack',
4431                'signal',
4432                'umount',
4433                'vfork']),
4434  'mprotect': set(['brk',
4435                   'capget',
4436                   'clone',
4437                   'fork',
4438                   'get_mempolicy',
4439                   'get_robust_list',
4440                   'getitimer',
4441                   'getpgid',
4442                   'getppid',
4443                   'getpriority',
4444                   'getrusage',
4445                   'getsid',
4446                   'ioprio_get',
4447                   'ioprio_set',
4448                   'keyctl',
4449                   'kill',
4450                   'madvise',
4451                   'migrate_pages',
4452                   'mincore',
4453                   'mlockall',
4454                   'move_pages',
4455                   'mq_timedreceive',
4456                   'mq_timedsend',
4457                   'mremap',
4458                   'msgrcv',
4459                   'munlock',
4460                   'munlockall',
4461                   'perf_event_open',
4462                   'personality',
4463                   'pkey_mprotect',
4464                   'prctl',
4465                   'prlimit64',
4466                   'ptrace',
4467                   'remap_file_pages',
4468                   'rt_sigaction',
4469                   'rt_sigprocmask',
4470                   'rt_sigtimedwait',
4471                   'sched_getaffinity',
4472                   'sched_getattr',
4473                   'sched_getparam',
4474                   'sched_getscheduler',
4475                   'sched_rr_get_interval',
4476                   'sched_setaffinity',
4477                   'sched_setattr',
4478                   'sched_setparam',
4479                   'sched_setscheduler',
4480                   'semtimedop',
4481                   'setitimer',
```

```
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'shmdt',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'mq_getsetattr': set(['accept4',
                       'acct',
                       'dup',
                       'dup3',
                       'epoll_create1',
                       'epoll_ctl',
                       'eventfd2',
                       'flock',
                       'memfd_create',
                       'mmap_pgoff',
                       'mq_notify',
                       'mq_open',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'open',
                       'open_by_handle_at',
                       'openat',
                       'perf_event_open',
                       'pipe2',
                       'remap_file_pages',
                       'setns',
                       'shmat',
                       'shmctl',
                       'shmdt',
                       'socket',
                       'socketpair',
                       'swapoff',
                       'swapon',
                       'uselib']),
 'mq_notify': set(['accept4',
                   'acct',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'flock',
                   'memfd_create',
                   'mmap_pgoff',
                   'mq_getsetattr',
                   'mq_open',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'remap_file_pages',
                   'rt_sigqueueinfo',
                   'rt_sigreturn',
                   'rt_sigtimedwait',
                   'rt_tgsigqueueinfo',
                   'setns',
                   'shmat',
                   'shmctl',
```



```
                   'shmdt',
                   'socket',
                   'socketpair',
                   'swapoff',
                   'swapon',
                   'tgkill',
                   'timer_create',
                   'tkill',
                   'uselib']),
 'mq_open': set(['accept4',
                 'acct',
                 'dup',
                 'dup3',
                 'epoll_create1',
                 'epoll_ctl',
                 'eventfd2',
                 'flock',
                 'getcwd',
                 'lookup_dcookie',
                 'memfd_create',
                 'mmap_pgoff',
                 'mq_unlink',
                 'open',
                 'open_by_handle_at',
                 'openat',
                 'perf_event_open',
                 'pipe2',
                 'pivot_root',
                 'quotactl',
                 'remap_file_pages',
                 'renameat2',
                 'rmdir',
                 'setns',
                 'shmat',
                 'shmctl',
                 'shmdt',
                 'socket',
                 'socketpair',
                 'swapoff',
                 'swapon',
                 'symlinkat',
                 'sysfs',
                 'unlink',
                 'unlinkat',
                 'unshare',
                 'uselib']),
 'mq_timedreceive': set(['accept4',
                         'acct',
                         'dup',
                         'dup3',
                         'epoll_create1',
                         'epoll_ctl',
                         'eventfd2',
                         'flock',
                         'memfd_create',
                         'mmap_pgoff',
                         'mq_getsetattr',
                         'mq_notify',
                         'mq_open',
                         'mq_timedsend',
                         'msgrcv',
                         'msgsnd',
                         'open',
                         'open_by_handle_at',
                         'openat',
                         'perf_event_open',
```

```
                        'pipe2',
                        'remap_file_pages',
                        'setns',
                        'shmat',
                        'shmctl',
                        'shmdt',
                        'socket',
                        'socketpair',
                        'swapoff',
                        'swapon',
                        'uselib']),
 'mq_timedsend': set(['accept4',
                      'acct',
                      'dup',
                      'dup3',
                      'epoll_create1',
                      'epoll_ctl',
                      'eventfd2',
                      'flock',
                      'memfd_create',
                      'mmap_pgoff',
                      'mq_getsetattr',
                      'mq_notify',
                      'mq_open',
                      'mq_timedreceive',
                      'open',
                      'open_by_handle_at',
                      'openat',
                      'perf_event_open',
                      'pipe2',
                      'remap_file_pages',
                      'setns',
                      'shmat',
                      'shmctl',
                      'shmdt',
                      'socket',
                      'socketpair',
                      'swapoff',
                      'swapon',
                      'uselib']),
 'mremap': set(['brk',
                'capget',
                'clone',
                'fork',
                'get_mempolicy',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'io_cancel',
                'io_destroy',
                'io_getevents',
                'io_setup',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'madvise',
                'mbind',
                'migrate_pages',
                'mincore',
                'mlockall',
                'modify_ldt',
```



```
                'move_pages',
                'mprotect',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'munlock',
                'munlockall',
                'perf_event_open',
                'personality',
                'pkey_mprotect',
                'prctl',
                'prlimit64',
                'ptrace',
                'remap_file_pages',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmdt',
                'sigaction',
                'sigaltstack',
                'signal',
                'swapoff',
                'umount',
                'vfork']),
 'msgctl': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioperm',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'msgget',
                'msgrcv',
                'msgsnd',
                'perf_event_open',
                'prctl',
                'prlimit64',
```

```
4746                 'ptrace',
4747                 'rt_sigaction',
4748                 'rt_sigprocmask',
4749                 'rt_sigtimedwait',
4750                 'sched_getaffinity',
4751                 'sched_getattr',
4752                 'sched_getparam',
4753                 'sched_getscheduler',
4754                 'sched_rr_get_interval',
4755                 'sched_setaffinity',
4756                 'sched_setattr',
4757                 'sched_setparam',
4758                 'sched_setscheduler',
4759                 'semctl',
4760                 'semget',
4761                 'semtimedop',
4762                 'setitimer',
4763                 'setns',
4764                 'setpgid',
4765                 'setpriority',
4766                 'setsid',
4767                 'shmat',
4768                 'shmctl',
4769                 'shmget',
4770                 'sigaction',
4771                 'sigaltstack',
4772                 'signal',
4773                 'umount',
4774                 'vfork']),
4775  'msgrcv': set(['mq_timedreceive', 'mq_timedsend', 'msgsnd']),
4776  'msgsnd': set(['mq_open',
4777                 'mq_unlink',
4778                 'msgctl',
4779                 'msgget',
4780                 'msgrcv',
4781                 'semctl',
4782                 'semget',
4783                 'semtimedop',
4784                 'setns',
4785                 'shmat',
4786                 'shmctl',
4787                 'shmget']),
4788  'msync': set(['brk',
4789                'get_mempolicy',
4790                'madvise',
4791                'mincore',
4792                'mlockall',
4793                'mprotect',
4794                'mremap',
4795                'munlock',
4796                'munlockall',
4797                'pkey_mprotect',
4798                'prctl',
4799                'remap_file_pages',
4800                'shmdt']),
4801  'munlock': set(['brk',
4802                  'capget',
4803                  'clone',
4804                  'fork',
4805                  'get_mempolicy',
4806                  'get_robust_list',
4807                  'getitimer',
4808                  'getpgid',
4809                  'getppid',
4810                  'getpriority',
4811                  'getrusage',
```



```
4812                 'getsid',
4813                 'ioprio_get',
4814                 'ioprio_set',
4815                 'keyctl',
4816                 'kill',
4817                 'madvise',
4818                 'migrate_pages',
4819                 'mincore',
4820                 'mlockall',
4821                 'move_pages',
4822                 'mprotect',
4823                 'mq_timedreceive',
4824                 'mq_timedsend',
4825                 'mremap',
4826                 'msgrcv',
4827                 'munlockall',
4828                 'perf_event_open',
4829                 'pkey_mprotect',
4830                 'prctl',
4831                 'prlimit64',
4832                 'ptrace',
4833                 'remap_file_pages',
4834                 'rt_sigaction',
4835                 'rt_sigprocmask',
4836                 'rt_sigtimedwait',
4837                 'sched_getaffinity',
4838                 'sched_getattr',
4839                 'sched_getparam',
4840                 'sched_getscheduler',
4841                 'sched_rr_get_interval',
4842                 'sched_setaffinity',
4843                 'sched_setattr',
4844                 'sched_setparam',
4845                 'sched_setscheduler',
4846                 'semtimedop',
4847                 'setitimer',
4848                 'setns',
4849                 'setpgid',
4850                 'setpriority',
4851                 'setsid',
4852                 'shmdt',
4853                 'sigaction',
4854                 'sigaltstack',
4855                 'signal',
4856                 'umount',
4857                 'vfork']),
4858  'munlockall': set(['brk',
4859                     'get_mempolicy',
4860                     'madvise',
4861                     'mincore',
4862                     'mlockall',
4863                     'mprotect',
4864                     'mremap',
4865                     'munlock',
4866                     'pkey_mprotect',
4867                     'prctl',
4868                     'remap_file_pages',
4869                     'shmdt']),
4870  'name_to_handle_at': set(['accept4',
4871                            'acct',
4872                            'dup',
4873                            'dup3',
4874                            'epoll_create1',
4875                            'epoll_ctl',
4876                            'eventfd2',
4877                            'flock',
```

```
                          'getcwd',
                          'lookup_dcookie',
                          'memfd_create',
                          'mmap_pgoff',
                          'mq_open',
                          'open',
                          'open_by_handle_at',
                          'openat',
                          'perf_event_open',
                          'pipe2',
                          'pivot_root',
                          'quotactl',
                          'remap_file_pages',
                          'setns',
                          'shmat',
                          'shmctl',
                          'shmdt',
                          'socket',
                          'socketpair',
                          'swapoff',
                          'swapon',
                          'syncfs',
                          'umount',
                          'unshare',
                          'uselib',
                          'ustat']),
 'nanosleep': set(['clock_nanosleep',
                   'epoll_wait',
                   'faccessat',
                   'fchmod',
                   'fchmodat',
                   'fchown',
                   'fchownat',
                   'fstat',
                   'ftruncate',
                   'futex',
                   'futimesat',
                   'inotify_add_watch',
                   'io_getevents',
                   'ioctl',
                   'linkat',
                   'memfd_create',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'mq_unlink',
                   'newfstat',
                   'poll',
                   'ppoll',
                   'pselect6',
                   'readlinkat',
                   'recvmmsg',
                   'rt_sigtimedwait',
                   'sched_rr_get_interval',
                   'select',
                   'semtimedop',
                   'sendfile',
                   'sendfile64',
                   'settimeofday',
                   'stime',
                   'swapoff',
                   'swapon',
                   'timer_gettime',
                   'timer_settime',
                   'timerfd_gettime',
```



```
                   'timerfd_settime',
                   'unlink',
                   'unlinkat',
                   'uselib',
                   'utime']),
 'newfstat': set(['fstat', 'newfstatat', 'newlstat', 'newstat']),
 'newfstatat': set(['fstat', 'newfstat', 'newlstat', 'newstat']),
 'newlstat': set(['fstat', 'newfstat', 'newfstatat', 'newstat']),
 'newstat': set(['fstat', 'newfstat', 'newfstatat', 'newlstat']),
 'newuname': set(['capget',
                  'clone',
                  'fork',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'personality',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'old_getrlimit': set(['prlimit64', 'setrlimit']),
 'old_readdir': set(['accept4',
                     'bind',
                     'bpf',
                     'connect',
                     'copy_file_range',
                     'epoll_ctl',
                     'epoll_wait',
                     'fallocate',
                     'fchdir',
```

```
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'olduname': set(['capget',
                  'clone',
                  'fork',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
```



```
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'ioperm',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'personality',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'open_by_handle_at': set(['accept4',
                           'acct',
                           'dup',
                           'dup3',
                           'epoll_create1',
                           'epoll_ctl',
                           'eventfd2',
                           'flock',
                           'getcwd',
                           'lookup_dcookie',
                           'memfd_create',
                           'mmap_pgoff',
                           'mq_open',
                           'open',
                           'openat',
                           'perf_event_open',
                           'pipe2',
                           'pivot_root',
                           'quotactl',
                           'remap_file_pages',
                           'setns',
                           'shmat',
                           'shmctl',
                           'shmdt',
```

```
                          'socket',
                          'socketpair',
                          'swapoff',
                          'swapon',
                          'unshare',
                          'uselib']),
 'perf_event_open': set(['accept4',
                         'acct',
                         'bind',
                         'bpf',
                         'brk',
                         'capget',
                         'clone',
                         'connect',
                         'copy_file_range',
                         'delete_module',
                         'dup',
                         'dup2',
                         'dup3',
                         'epoll_create1',
                         'epoll_ctl',
                         'epoll_wait',
                         'eventfd2',
                         'exit_group',
                         'faccessat',
                         'fallocate',
                         'fchdir',
                         'fchmod',
                         'fchmodat',
                         'fchown',
                         'fchownat',
                         'fcntl',
                         'fcntl64',
                         'fdatasync',
                         'fgetxattr',
                         'finit_module',
                         'flistxattr',
                         'flock',
                         'fork',
                         'fremovexattr',
                         'fsetxattr',
                         'fstatfs',
                         'fstatfs64',
                         'fsync',
                         'ftruncate',
                         'futimesat',
                         'get_curr_temp',
                         'get_mempolicy',
                         'get_robust_list',
                         'get_trip_temp',
                         'getcwd',
                         'getdents',
                         'getdents64',
                         'getgroups',
                         'getgroups16',
                         'getitimer',
                         'getpeername',
                         'getpgid',
                         'getppid',
                         'getpriority',
                         'getrusage',
                         'getsid',
                         'getsockname',
                         'getsockopt',
                         'init_module',
                         'inotify_add_watch',
```



```
                         'inotify_init1',
                         'inotify_rm_watch',
                         'io_cancel',
                         'io_destroy',
                         'io_getevents',
                         'io_setup',
                         'io_submit',
                         'ioctl',
                         'ioperm',
                         'ioprio_get',
                         'ioprio_set',
                         'kexec_load',
                         'keyctl',
                         'kill',
                         'linkat',
                         'listen',
                         'llseek',
                         'lookup_dcookie',
                         'lseek',
                         'madvise',
                         'mbind',
                         'memfd_create',
                         'migrate_pages',
                         'mincore',
                         'mkdirat',
                         'mknodat',
                         'mlockall',
                         'mmap_pgoff',
                         'modify_ldt',
                         'move_pages',
                         'mprotect',
                         'mq_getsetattr',
                         'mq_notify',
                         'mq_open',
                         'mq_timedreceive',
                         'mq_timedsend',
                         'mq_unlink',
                         'mremap',
                         'msgctl',
                         'msgrcv',
                         'msgsnd',
                         'munlock',
                         'munlockall',
                         'old_readdir',
                         'open',
                         'open_by_handle_at',
                         'openat',
                         'pipe2',
                         'pivot_root',
                         'pkey_mprotect',
                         'prctl',
                         'pread64',
                         'preadv',
                         'preadv2',
                         'preadv64',
                         'preadv64v2',
                         'prlimit64',
                         'ptrace',
                         'pwrite64',
                         'pwritev',
                         'pwritev2',
                         'pwritev64',
                         'pwritev64v2',
                         'quotactl',
                         'read',
                         'readahead',
```

```
5274                        'readlinkat',
5275                        'readv',
5276                        'reboot',
5277                        'recvfrom',
5278                        'remap_file_pages',
5279                        'renameat2',
5280                        'request_key',
5281                        'rmdir',
5282                        'rt_sigaction',
5283                        'rt_sigprocmask',
5284                        'rt_sigtimedwait',
5285                        'sched_getaffinity',
5286                        'sched_getattr',
5287                        'sched_getparam',
5288                        'sched_getscheduler',
5289                        'sched_rr_get_interval',
5290                        'sched_setaffinity',
5291                        'sched_setattr',
5292                        'sched_setparam',
5293                        'sched_setscheduler',
5294                        'sched_yield',
5295                        'semctl',
5296                        'semtimedop',
5297                        'sendfile',
5298                        'sendfile64',
5299                        'sendto',
5300                        'set_trip_temp',
5301                        'setgid',
5302                        'setgroups',
5303                        'setgroups16',
5304                        'setitimer',
5305                        'setns',
5306                        'setpgid',
5307                        'setpriority',
5308                        'setregid',
5309                        'setresgid',
5310                        'setresuid',
5311                        'setreuid',
5312                        'setsid',
5313                        'setsockopt',
5314                        'setuid',
5315                        'shmat',
5316                        'shmctl',
5317                        'shmdt',
5318                        'shutdown',
5319                        'sigaction',
5320                        'sigaltstack',
5321                        'signal',
5322                        'signalfd4',
5323                        'socket',
5324                        'socketpair',
5325                        'splice',
5326                        'swapoff',
5327                        'swapon',
5328                        'symlinkat',
5329                        'sync_file_range',
5330                        'syncfs',
5331                        'tee',
5332                        'timer_create',
5333                        'timer_delete',
5334                        'timer_getoverrun',
5335                        'timer_gettime',
5336                        'timer_settime',
5337                        'timerfd_create',
5338                        'timerfd_gettime',
5339                        'timerfd_settime',
```



```
5340                        'umount',
5341                        'unlink',
5342                        'unlinkat',
5343                        'unshare',
5344                        'uselib',
5345                        'ustat',
5346                        'utime',
5347                        'utimensat',
5348                        'vfork',
5349                        'vmsplice',
5350                        'write',
5351                        'writev']),
5352  'pivot_root': set(['accept4',
5353                     'acct',
5354                     'dup',
5355                     'dup3',
5356                     'epoll_create1',
5357                     'epoll_ctl',
5358                     'eventfd2',
5359                     'flock',
5360                     'ftruncate',
5361                     'getcwd',
5362                     'linkat',
5363                     'lookup_dcookie',
5364                     'memfd_create',
5365                     'mkdirat',
5366                     'mknodat',
5367                     'mmap_pgoff',
5368                     'mq_open',
5369                     'mq_unlink',
5370                     'open',
5371                     'open_by_handle_at',
5372                     'openat',
5373                     'perf_event_open',
5374                     'pipe2',
5375                     'quotactl',
5376                     'remap_file_pages',
5377                     'renameat2',
5378                     'rmdir',
5379                     'setns',
5380                     'shmat',
5381                     'shmctl',
5382                     'shmdt',
5383                     'socket',
5384                     'socketpair',
5385                     'swapoff',
5386                     'swapon',
5387                     'symlinkat',
5388                     'umount',
5389                     'unlink',
5390                     'unlinkat',
5391                     'unshare',
5392                     'uselib']),
5393  'pkey_alloc': set(['brk',
5394                     'get_mempolicy',
5395                     'getrusage',
5396                     'io_cancel',
5397                     'io_destroy',
5398                     'io_getevents',
5399                     'io_setup',
5400                     'mbind',
5401                     'migrate_pages',
5402                     'mincore',
5403                     'modify_ldt',
5404                     'move_pages',
5405                     'mremap',
```

```
                    'pkey_free',
                    'prctl',
                    'remap_file_pages',
                    'shmdt',
                    'swapoff']),
 'pkey_mprotect': set(['brk',
                       'capget',
                       'clone',
                       'fork',
                       'get_mempolicy',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'madvise',
                       'migrate_pages',
                       'mincore',
                       'mlockall',
                       'move_pages',
                       'mprotect',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'mremap',
                       'msgrcv',
                       'munlock',
                       'munlockall',
                       'perf_event_open',
                       'personality',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'remap_file_pages',
                       'rt_sigaction',
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getattr',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setattr',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'shmdt',
                       'sigaction',
                       'sigaltstack',
                       'signal',
                       'umount',
                       'vfork']),
 'poll': set(['ppoll']),
 'ppoll': set(['capget',
               'clock_nanosleep',
```



```
               'clone',
               'epoll_wait',
               'faccessat',
               'fchmod',
               'fchmodat',
               'fchown',
               'fchownat',
               'fork',
               'fstat',
               'ftruncate',
               'futex',
               'futimesat',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'inotify_add_watch',
               'io_getevents',
               'ioctl',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'linkat',
               'memfd_create',
               'migrate_pages',
               'move_pages',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'mq_unlink',
               'msgrcv',
               'nanosleep',
               'newfstat',
               'perf_event_open',
               'personality',
               'poll',
               'prctl',
               'prlimit64',
               'pselect6',
               'ptrace',
               'readlinkat',
               'recvmmsg',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
               'select',
               'semtimedop',
               'sendfile',
               'sendfile64',
               'setitimer',
               'setns',
               'setpgid',
```

```
5538                'setpriority',
5539                'setsid',
5540                'settimeofday',
5541                'sigaction',
5542                'sigaltstack',
5543                'signal',
5544                'stime',
5545                'swapoff',
5546                'swapon',
5547                'timer_gettime',
5548                'timer_settime',
5549                'timerfd_gettime',
5550                'timerfd_settime',
5551                'umount',
5552                'unlink',
5553                'unlinkat',
5554                'uselib',
5555                'utime',
5556                'vfork']),
5557  'prctl': set(['brk',
5558                'capget',
5559                'clone',
5560                'fork',
5561                'get_mempolicy',
5562                'get_robust_list',
5563                'getitimer',
5564                'getpgid',
5565                'getppid',
5566                'getpriority',
5567                'getrusage',
5568                'getsid',
5569                'io_cancel',
5570                'io_destroy',
5571                'io_getevents',
5572                'io_setup',
5573                'ioprio_get',
5574                'ioprio_set',
5575                'keyctl',
5576                'kill',
5577                'mbind',
5578                'migrate_pages',
5579                'mincore',
5580                'modify_ldt',
5581                'move_pages',
5582                'mq_timedreceive',
5583                'mq_timedsend',
5584                'mremap',
5585                'msgrcv',
5586                'perf_event_open',
5587                'personality',
5588                'prlimit64',
5589                'ptrace',
5590                'remap_file_pages',
5591                'rt_sigaction',
5592                'rt_sigprocmask',
5593                'rt_sigtimedwait',
5594                'sched_getaffinity',
5595                'sched_getattr',
5596                'sched_getparam',
5597                'sched_getscheduler',
5598                'sched_rr_get_interval',
5599                'sched_setaffinity',
5600                'sched_setattr',
5601                'sched_setparam',
5602                'sched_setscheduler',
5603                'semtimedop',
```



```
5604                'setitimer',
5605                'setns',
5606                'setpgid',
5607                'setpriority',
5608                'setresuid',
5609                'setreuid',
5610                'setsid',
5611                'setuid',
5612                'shmdt',
5613                'sigaction',
5614                'sigaltstack',
5615                'signal',
5616                'swapoff',
5617                'umount',
5618                'vfork']),
5619  'pread64': set(['accept4',
5620                  'acct',
5621                  'bind',
5622                  'bpf',
5623                  'connect',
5624                  'copy_file_range',
5625                  'dup',
5626                  'dup3',
5627                  'epoll_create1',
5628                  'epoll_ctl',
5629                  'epoll_wait',
5630                  'eventfd2',
5631                  'fallocate',
5632                  'fchdir',
5633                  'fchmod',
5634                  'fchown',
5635                  'fcntl',
5636                  'fcntl64',
5637                  'fdatasync',
5638                  'fgetxattr',
5639                  'flistxattr',
5640                  'flock',
5641                  'fremovexattr',
5642                  'fsetxattr',
5643                  'fstatfs',
5644                  'fstatfs64',
5645                  'fsync',
5646                  'ftruncate',
5647                  'futimesat',
5648                  'getdents',
5649                  'getdents64',
5650                  'getpeername',
5651                  'getsockname',
5652                  'getsockopt',
5653                  'inotify_add_watch',
5654                  'inotify_rm_watch',
5655                  'ioctl',
5656                  'listen',
5657                  'llseek',
5658                  'lseek',
5659                  'memfd_create',
5660                  'mmap_pgoff',
5661                  'mq_getsetattr',
5662                  'mq_notify',
5663                  'mq_open',
5664                  'mq_timedreceive',
5665                  'mq_timedsend',
5666                  'old_readdir',
5667                  'open',
5668                  'open_by_handle_at',
5669                  'openat',
```

```
                'perf_event_open',
                'pipe2',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'preadv': set(['accept4',
                'acct',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'eventfd2',
                'flock',
                'memfd_create',
                'mmap_pgoff',
                'mq_open',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'remap_file_pages',
                'setns',
                'shmat',
                'shmctl',
                'shmdt',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'uselib']),
```



```
 'preadv2': set(['accept4',
                 'acct',
                 'dup',
                 'dup3',
                 'epoll_create1',
                 'epoll_ctl',
                 'eventfd2',
                 'flock',
                 'memfd_create',
                 'mmap_pgoff',
                 'mq_open',
                 'open',
                 'open_by_handle_at',
                 'openat',
                 'perf_event_open',
                 'pipe2',
                 'remap_file_pages',
                 'setns',
                 'shmat',
                 'shmctl',
                 'shmdt',
                 'socket',
                 'socketpair',
                 'swapoff',
                 'swapon',
                 'uselib']),
 'preadv64': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'uselib']),
 'preadv64v2': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_open',
                    'open',
                    'open_by_handle_at',
                    'openat',
```

```
                    'perf_event_open',
                    'pipe2',
                    'remap_file_pages',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'uselib']),
 'prlimit64': set(['capget',
                   'capset',
                   'clone',
                   'epoll_create1',
                   'faccessat',
                   'fork',
                   'get_robust_list',
                   'getgroups',
                   'getgroups16',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getresgid',
                   'getresgid16',
                   'getresuid',
                   'getresuid16',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'old_getrlimit',
                   'perf_event_open',
                   'prctl',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setfsgid',
                   'setfsuid',
                   'setgid',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setregid',
                   'setresgid',
```



```
                    'setresuid',
                    'setreuid',
                    'setrlimit',
                    'setsid',
                    'setuid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'unshare',
                    'vfork']),
 'pselect6': set(['capget',
                  'clock_nanosleep',
                  'clone',
                  'epoll_wait',
                  'faccessat',
                  'fchmod',
                  'fchmodat',
                  'fchown',
                  'fchownat',
                  'fork',
                  'fstat',
                  'ftruncate',
                  'futex',
                  'futimesat',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'inotify_add_watch',
                  'io_getevents',
                  'ioctl',
                  'ioperm',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'linkat',
                  'memfd_create',
                  'migrate_pages',
                  'move_pages',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mq_unlink',
                  'msgrcv',
                  'nanosleep',
                  'newfstat',
                  'perf_event_open',
                  'poll',
                  'ppoll',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'readlinkat',
                  'recvmmsg',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
```

```
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'select',
                  'semtimedop',
                  'sendfile',
                  'sendfile64',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'settimeofday',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'stime',
                  'swapoff',
                  'swapon',
                  'timer_gettime',
                  'timer_settime',
                  'timerfd_gettime',
                  'timerfd_settime',
                  'umount',
                  'unlink',
                  'unlinkat',
                  'uselib',
                  'utime',
                  'vfork']),
 'ptrace': set(['capget',
                'clone',
                'epoll_wait',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'pause',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigsuspend',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
```



```
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setresuid',
                'setreuid',
                'setsid',
                'setuid',
                'sigaction',
                'sigaltstack',
                'signal',
                'sigsuspend',
                'umount',
                'vfork']),
 'pwrite64': set(['accept4',
                  'acct',
                  'bind',
                  'bpf',
                  'connect',
                  'copy_file_range',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'epoll_wait',
                  'eventfd2',
                  'fallocate',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
                  'fremovexattr',
                  'fsetxattr',
                  'fstatfs',
                  'fstatfs64',
                  'fsync',
                  'ftruncate',
                  'futimesat',
                  'getdents',
                  'getdents64',
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'inotify_rm_watch',
                  'ioctl',
                  'listen',
                  'llseek',
                  'lseek',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_open',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'old_readdir',
```

```
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pread64',
                  'preadv',
                  'preadv2',
                  'preadv64',
                  'preadv64v2',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'remap_file_pages',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setns',
                  'setsockopt',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'shutdown',
                  'signalfd4',
                  'socket',
                  'socketpair',
                  'splice',
                  'swapoff',
                  'swapon',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'uselib',
                  'utime',
                  'utimensat',
                  'vmsplice',
                  'write',
                  'writev']),
 'pwritev': set(['accept4',
                 'acct',
                 'dup',
                 'dup3',
                 'epoll_create1',
                 'epoll_ctl',
                 'eventfd2',
                 'flock',
                 'memfd_create',
                 'mmap_pgoff',
                 'mq_open',
                 'open',
                 'open_by_handle_at',
                 'openat',
                 'perf_event_open',
                 'pipe2',
                 'remap_file_pages',
                 'setns',
                 'shmat',
                 'shmctl',
                 'shmdt',
                 'socket',
                 'socketpair',
```



```
                 'swapoff',
                 'swapon',
                 'uselib']),
 'pwritev2': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'uselib']),
 'pwritev64': set(['accept4',
                   'acct',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'flock',
                   'memfd_create',
                   'mmap_pgoff',
                   'mq_open',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'remap_file_pages',
                   'setns',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'socket',
                   'socketpair',
                   'swapoff',
                   'swapon',
                   'uselib']),
 'pwritev64v2': set(['accept4',
                     'acct',
                     'dup',
                     'dup3',
                     'epoll_create1',
                     'epoll_ctl',
                     'eventfd2',
                     'flock',
                     'memfd_create',
                     'mmap_pgoff',
                     'mq_open',
```

```
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'remap_file_pages',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'uselib']),
 'quotactl': set(['acct',
                  'mq_open',
                  'mq_unlink',
                  'open',
                  'openat',
                  'renameat2',
                  'rmdir',
                  'swapoff',
                  'swapon',
                  'symlinkat',
                  'syncfs',
                  'sysfs',
                  'umount',
                  'unlink',
                  'unlinkat',
                  'uselib',
                  'ustat']),
 'read': set(['accept4',
              'bind',
              'bpf',
              'connect',
              'copy_file_range',
              'epoll_ctl',
              'epoll_wait',
              'fallocate',
              'fchdir',
              'fchmod',
              'fchown',
              'fcntl',
              'fcntl64',
              'fdatasync',
              'fgetxattr',
              'flistxattr',
              'flock',
              'fremovexattr',
              'fsetxattr',
              'fstatfs',
              'fstatfs64',
              'fsync',
              'ftruncate',
              'futimesat',
              'getdents',
              'getdents64',
              'getpeername',
              'getsockname',
              'getsockopt',
              'inotify_add_watch',
              'inotify_rm_watch',
              'ioctl',
              'listen',
              'llseek',
```



```
              'lseek',
              'mq_getsetattr',
              'mq_notify',
              'mq_timedreceive',
              'mq_timedsend',
              'old_readdir',
              'perf_event_open',
              'pread64',
              'preadv',
              'preadv2',
              'preadv64',
              'preadv64v2',
              'pwrite64',
              'pwritev',
              'pwritev2',
              'pwritev64',
              'pwritev64v2',
              'readahead',
              'readv',
              'recvfrom',
              'sendfile',
              'sendfile64',
              'sendto',
              'setsockopt',
              'shutdown',
              'signalfd4',
              'splice',
              'sync_file_range',
              'syncfs',
              'tee',
              'utime',
              'utimensat',
              'vmsplice',
              'write',
              'writev']),
 'readahead': set(['accept4',
                   'acct',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'epoll_wait',
                   'eventfd2',
                   'faccessat',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchmodat',
                   'fchown',
                   'fchownat',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'fgetxattr',
                   'flistxattr',
                   'flock',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
```

```
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'linkat',
                    'listen',
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'mq_unlink',
                    'old_readdir',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readlinkat',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setns',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'signalfd4',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'unlink',
                    'unlinkat',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
```



```
                    'write',
                    'writev']),
 'readlinkat': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'getcwd',
                    'lookup_dcookie',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_open',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pivot_root',
                    'quotactl',
                    'remap_file_pages',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'unshare',
                    'uselib']),
 'reboot': set(['acct', 'perf_event_open', 'setns']),
 'recvfrom': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_getsetattr',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'uselib']),
 'recvmmsg': set(['accept4',
                  'bind',
                  'clock_nanosleep',
                  'connect',
                  'epoll_wait',
```

```
                  'faccessat',
                  'fchmod',
                  'fchmodat',
                  'fchown',
                  'fchownat',
                  'fstat',
                  'ftruncate',
                  'futex',
                  'futimesat',
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'io_getevents',
                  'ioctl',
                  'linkat',
                  'listen',
                  'memfd_create',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mq_unlink',
                  'nanosleep',
                  'newfstat',
                  'poll',
                  'ppoll',
                  'pselect6',
                  'readlinkat',
                  'recvfrom',
                  'recvmsg',
                  'rt_sigtimedwait',
                  'sched_rr_get_interval',
                  'select',
                  'semtimedop',
                  'sendfile',
                  'sendfile64',
                  'sendmmsg',
                  'sendmsg',
                  'sendto',
                  'setsockopt',
                  'settimeofday',
                  'shutdown',
                  'stime',
                  'swapoff',
                  'swapon',
                  'timer_gettime',
                  'timer_settime',
                  'timerfd_gettime',
                  'timerfd_settime',
                  'unlink',
                  'unlinkat',
                  'uselib',
                  'utime']),
  'recvmsg': set(['accept4',
                  'bind',
                  'connect',
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'listen',
                  'recvfrom',
                  'recvmmsg',
                  'sendmmsg',
                  'sendmsg',
                  'sendto',
```



```
                  'setsockopt',
                  'shutdown']),
  'remap_file_pages': set(['brk',
                           'get_mempolicy',
                           'madvise',
                           'mincore',
                           'mlockall',
                           'mprotect',
                           'mremap',
                           'munlock',
                           'munlockall',
                           'pkey_mprotect',
                           'prctl',
                           'shmdt']),
  'removexattr': set(['accept4',
                      'acct',
                      'dup',
                      'dup3',
                      'epoll_create1',
                      'epoll_ctl',
                      'eventfd2',
                      'flock',
                      'getcwd',
                      'lookup_dcookie',
                      'memfd_create',
                      'mmap_pgoff',
                      'mq_open',
                      'open',
                      'open_by_handle_at',
                      'openat',
                      'perf_event_open',
                      'pipe2',
                      'pivot_root',
                      'quotactl',
                      'remap_file_pages',
                      'setns',
                      'shmat',
                      'shmctl',
                      'shmdt',
                      'socket',
                      'socketpair',
                      'swapoff',
                      'swapon',
                      'unshare',
                      'uselib']),
  'renameat2': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'ftruncate',
                    'getcwd',
                    'linkat',
                    'lookup_dcookie',
                    'memfd_create',
                    'mkdirat',
                    'mknodat',
                    'mmap_pgoff',
                    'mq_open',
                    'mq_unlink',
                    'open',
                    'open_by_handle_at',
                    'openat',
```

```
                    'perf_event_open',
                    'pipe2',
                    'pivot_root',
                    'quotactl',
                    'remap_file_pages',
                    'rmdir',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'symlinkat',
                    'sysfs',
                    'unlink',
                    'unlinkat',
                    'unshare',
                    'uselib']),
 'rmdir': set(['accept4',
               'acct',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'eventfd2',
               'flock',
               'ftruncate',
               'getcwd',
               'linkat',
               'lookup_dcookie',
               'memfd_create',
               'mkdirat',
               'mknodat',
               'mmap_pgoff',
               'mq_open',
               'mq_unlink',
               'open',
               'open_by_handle_at',
               'openat',
               'perf_event_open',
               'pipe2',
               'pivot_root',
               'quotactl',
               'remap_file_pages',
               'renameat2',
               'setns',
               'shmat',
               'shmctl',
               'shmdt',
               'socket',
               'socketpair',
               'swapoff',
               'swapon',
               'symlinkat',
               'unlink',
               'unlinkat',
               'unshare',
               'uselib']),
 'rt_sigqueueinfo': set(['kill',
                         'rt_sigreturn',
                         'rt_sigtimedwait',
                         'rt_tgsigqueueinfo',
                         'tgkill',
                         'timer_create',
```



```
                         'tkill']),
 'rt_sigreturn': set(['capget',
                      'clone',
                      'fork',
                      'get_robust_list',
                      'getitimer',
                      'getpgid',
                      'getppid',
                      'getpriority',
                      'getrusage',
                      'getsid',
                      'ioperm',
                      'ioprio_get',
                      'ioprio_set',
                      'keyctl',
                      'kill',
                      'migrate_pages',
                      'move_pages',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'msgrcv',
                      'perf_event_open',
                      'prctl',
                      'prlimit64',
                      'ptrace',
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setsid',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'umount',
                      'vfork']),
 'rt_sigtimedwait': set(['capget',
                         'clone',
                         'fork',
                         'get_robust_list',
                         'getitimer',
                         'getpgid',
                         'getppid',
                         'getpriority',
                         'getrusage',
                         'getsid',
                         'ioperm',
                         'ioprio_get',
                         'ioprio_set',
                         'keyctl',
                         'kill',
                         'migrate_pages',
                         'move_pages',
                         'mq_timedreceive',
```

```
                        'mq_timedsend',
                        'msgrcv',
                        'perf_event_open',
                        'prctl',
                        'prlimit64',
                        'ptrace',
                        'rt_sigaction',
                        'rt_sigprocmask',
                        'rt_sigqueueinfo',
                        'rt_sigreturn',
                        'rt_tgsigqueueinfo',
                        'sched_getaffinity',
                        'sched_getattr',
                        'sched_getparam',
                        'sched_getscheduler',
                        'sched_rr_get_interval',
                        'sched_setaffinity',
                        'sched_setattr',
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setsid',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'tgkill',
                        'timer_create',
                        'tkill',
                        'umount',
                        'vfork']),
 'rt_tgsigqueueinfo': set(['kill',
                           'rt_sigqueueinfo',
                           'rt_sigreturn',
                           'rt_sigtimedwait',
                           'tgkill',
                           'timer_create',
                           'tkill']),
 'sched_getattr': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioperm',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'migrate_pages',
                       'move_pages',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'msgrcv',
                       'perf_event_open',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'rt_sigaction',
```



```
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setattr',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'sigaction',
                       'sigaltstack',
                       'signal',
                       'umount',
                       'vfork']),
 'sched_getparam': set(['capget',
                        'clone',
                        'fork',
                        'get_robust_list',
                        'getitimer',
                        'getpgid',
                        'getppid',
                        'getpriority',
                        'getrusage',
                        'getsid',
                        'ioprio_get',
                        'ioprio_set',
                        'keyctl',
                        'kill',
                        'migrate_pages',
                        'move_pages',
                        'mq_timedreceive',
                        'mq_timedsend',
                        'msgrcv',
                        'perf_event_open',
                        'prctl',
                        'prlimit64',
                        'ptrace',
                        'rt_sigaction',
                        'rt_sigprocmask',
                        'rt_sigtimedwait',
                        'sched_getaffinity',
                        'sched_getattr',
                        'sched_getscheduler',
                        'sched_rr_get_interval',
                        'sched_setaffinity',
                        'sched_setattr',
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setsid',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'umount',
                        'vfork']),
```

```
6858  'sched_getscheduler': set(['capget',
6859                             'clone',
6860                             'fork',
6861                             'get_robust_list',
6862                             'getitimer',
6863                             'getpgid',
6864                             'getppid',
6865                             'getpriority',
6866                             'getrusage',
6867                             'getsid',
6868                             'ioprio_get',
6869                             'ioprio_set',
6870                             'keyctl',
6871                             'kill',
6872                             'migrate_pages',
6873                             'move_pages',
6874                             'mq_timedreceive',
6875                             'mq_timedsend',
6876                             'msgrcv',
6877                             'perf_event_open',
6878                             'prctl',
6879                             'prlimit64',
6880                             'ptrace',
6881                             'rt_sigaction',
6882                             'rt_sigprocmask',
6883                             'rt_sigtimedwait',
6884                             'sched_getaffinity',
6885                             'sched_getattr',
6886                             'sched_getparam',
6887                             'sched_rr_get_interval',
6888                             'sched_setaffinity',
6889                             'sched_setattr',
6890                             'sched_setparam',
6891                             'sched_setscheduler',
6892                             'semtimedop',
6893                             'setitimer',
6894                             'setns',
6895                             'setpgid',
6896                             'setpriority',
6897                             'setsid',
6898                             'sigaction',
6899                             'sigaltstack',
6900                             'signal',
6901                             'umount',
6902                             'vfork']),
6903  'sched_setaffinity': set(['capget',
6904                            'capset',
6905                            'clone',
6906                            'epoll_create1',
6907                            'faccessat',
6908                            'fork',
6909                            'get_robust_list',
6910                            'getgroups',
6911                            'getgroups16',
6912                            'getitimer',
6913                            'getpgid',
6914                            'getppid',
6915                            'getpriority',
6916                            'getresgid',
6917                            'getresgid16',
6918                            'getresuid',
6919                            'getresuid16',
6920                            'getrusage',
6921                            'getsid',
6922                            'ioprio_get',
6923                            'ioprio_set',
```



```
6924                            'keyctl',
6925                            'kill',
6926                            'migrate_pages',
6927                            'move_pages',
6928                            'mq_timedreceive',
6929                            'mq_timedsend',
6930                            'msgrcv',
6931                            'perf_event_open',
6932                            'prctl',
6933                            'prlimit64',
6934                            'ptrace',
6935                            'rt_sigaction',
6936                            'rt_sigprocmask',
6937                            'rt_sigtimedwait',
6938                            'sched_getaffinity',
6939                            'sched_getattr',
6940                            'sched_getparam',
6941                            'sched_getscheduler',
6942                            'sched_rr_get_interval',
6943                            'sched_setattr',
6944                            'sched_setparam',
6945                            'sched_setscheduler',
6946                            'semtimedop',
6947                            'setfsgid',
6948                            'setfsuid',
6949                            'setgid',
6950                            'setitimer',
6951                            'setns',
6952                            'setpgid',
6953                            'setpriority',
6954                            'setregid',
6955                            'setresgid',
6956                            'setresuid',
6957                            'setreuid',
6958                            'setsid',
6959                            'setuid',
6960                            'sigaction',
6961                            'sigaltstack',
6962                            'signal',
6963                            'umount',
6964                            'unshare',
6965                            'vfork']),
6966  'sched_setattr': set(['capget',
6967                        'clone',
6968                        'fork',
6969                        'get_robust_list',
6970                        'getitimer',
6971                        'getpgid',
6972                        'getppid',
6973                        'getpriority',
6974                        'getrusage',
6975                        'getsid',
6976                        'ioperm',
6977                        'ioprio_get',
6978                        'ioprio_set',
6979                        'keyctl',
6980                        'kill',
6981                        'migrate_pages',
6982                        'move_pages',
6983                        'mq_timedreceive',
6984                        'mq_timedsend',
6985                        'msgrcv',
6986                        'perf_event_open',
6987                        'prctl',
6988                        'prlimit64',
6989                        'ptrace',
```

```
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setsid',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'umount',
                      'vfork']),
 'select': set(['capget',
                'clock_nanosleep',
                'clone',
                'dup2',
                'dup3',
                'epoll_wait',
                'faccessat',
                'fchmod',
                'fchmodat',
                'fchown',
                'fchownat',
                'fork',
                'fstat',
                'ftruncate',
                'futex',
                'futimesat',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'inotify_add_watch',
                'io_getevents',
                'ioctl',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'linkat',
                'memfd_create',
                'migrate_pages',
                'move_pages',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'msgrcv',
                'nanosleep',
                'newfstat',
                'perf_event_open',
                'personality',
```



```
                'poll',
                'ppoll',
                'prctl',
                'prlimit64',
                'pselect6',
                'ptrace',
                'readlinkat',
                'recvmmsg',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'sendfile',
                'sendfile64',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'settimeofday',
                'sigaction',
                'sigaltstack',
                'signal',
                'stime',
                'swapoff',
                'swapon',
                'timer_gettime',
                'timer_settime',
                'timerfd_gettime',
                'timerfd_settime',
                'umount',
                'unlink',
                'unlinkat',
                'unshare',
                'uselib',
                'utime',
                'vfork']),
 'semctl': set(['semtimedop']),
 'semtimedop': set(['accept4',
                    'acct',
                    'bpf',
                    'brk',
                    'capget',
                    'clock_nanosleep',
                    'clone',
                    'delete_module',
                    'dup',
                    'dup2',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'epoll_wait',
                    'eventfd2',
                    'exit_group',
                    'faccessat',
                    'fchmod',
                    'fchmodat',
```

```
7122                    'fchown',
7123                    'fchownat',
7124                    'finit_module',
7125                    'flock',
7126                    'fork',
7127                    'fstat',
7128                    'ftruncate',
7129                    'futex',
7130                    'futimesat',
7131                    'get_curr_temp',
7132                    'get_mempolicy',
7133                    'get_robust_list',
7134                    'get_trip_temp',
7135                    'getcwd',
7136                    'getgroups',
7137                    'getgroups16',
7138                    'getitimer',
7139                    'getpgid',
7140                    'getppid',
7141                    'getpriority',
7142                    'getrusage',
7143                    'getsid',
7144                    'init_module',
7145                    'inotify_add_watch',
7146                    'inotify_init1',
7147                    'inotify_rm_watch',
7148                    'io_cancel',
7149                    'io_destroy',
7150                    'io_getevents',
7151                    'io_setup',
7152                    'io_submit',
7153                    'ioctl',
7154                    'ioprio_get',
7155                    'ioprio_set',
7156                    'kexec_load',
7157                    'keyctl',
7158                    'kill',
7159                    'linkat',
7160                    'lookup_dcookie',
7161                    'madvise',
7162                    'mbind',
7163                    'memfd_create',
7164                    'migrate_pages',
7165                    'mincore',
7166                    'mkdirat',
7167                    'mknodat',
7168                    'mlockall',
7169                    'mmap_pgoff',
7170                    'modify_ldt',
7171                    'move_pages',
7172                    'mprotect',
7173                    'mq_getsetattr',
7174                    'mq_notify',
7175                    'mq_open',
7176                    'mq_timedreceive',
7177                    'mq_timedsend',
7178                    'mq_unlink',
7179                    'mremap',
7180                    'msgctl',
7181                    'msgrcv',
7182                    'msgsnd',
7183                    'munlock',
7184                    'munlockall',
7185                    'nanosleep',
7186                    'newfstat',
7187                    'open',
```



```
7188                    'open_by_handle_at',
7189                    'openat',
7190                    'perf_event_open',
7191                    'pipe2',
7192                    'pivot_root',
7193                    'pkey_mprotect',
7194                    'poll',
7195                    'ppoll',
7196                    'prctl',
7197                    'prlimit64',
7198                    'pselect6',
7199                    'ptrace',
7200                    'quotactl',
7201                    'readahead',
7202                    'readlinkat',
7203                    'reboot',
7204                    'recvmmsg',
7205                    'remap_file_pages',
7206                    'renameat2',
7207                    'request_key',
7208                    'rmdir',
7209                    'rt_sigaction',
7210                    'rt_sigprocmask',
7211                    'rt_sigtimedwait',
7212                    'sched_getaffinity',
7213                    'sched_getattr',
7214                    'sched_getparam',
7215                    'sched_getscheduler',
7216                    'sched_rr_get_interval',
7217                    'sched_setaffinity',
7218                    'sched_setattr',
7219                    'sched_setparam',
7220                    'sched_setscheduler',
7221                    'sched_yield',
7222                    'select',
7223                    'semctl',
7224                    'sendfile',
7225                    'sendfile64',
7226                    'set_trip_temp',
7227                    'setgid',
7228                    'setgroups',
7229                    'setgroups16',
7230                    'setitimer',
7231                    'setns',
7232                    'setpgid',
7233                    'setpriority',
7234                    'setregid',
7235                    'setresgid',
7236                    'setresuid',
7237                    'setreuid',
7238                    'setsid',
7239                    'settimeofday',
7240                    'setuid',
7241                    'shmat',
7242                    'shmctl',
7243                    'shmdt',
7244                    'sigaction',
7245                    'sigaltstack',
7246                    'signal',
7247                    'socket',
7248                    'socketpair',
7249                    'splice',
7250                    'stime',
7251                    'swapoff',
7252                    'swapon',
7253                    'symlinkat',
```

```
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'timer_create',
                    'timer_delete',
                    'timer_getoverrun',
                    'timer_gettime',
                    'timer_settime',
                    'timerfd_create',
                    'timerfd_gettime',
                    'timerfd_settime',
                    'umount',
                    'unlink',
                    'unlinkat',
                    'unshare',
                    'uselib',
                    'ustat',
                    'utime',
                    'vfork',
                    'vmsplice']),
 'sendfile': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'uselib']),
 'sendfile64': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_open',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'remap_file_pages',
                    'setns',
                    'shmat',
                    'shmctl',
```



```
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'uselib']),
 'sendmmsg': set(['accept4',
                  'bind',
                  'connect',
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'listen',
                  'recvfrom',
                  'recvmmsg',
                  'recvmsg',
                  'sendmsg',
                  'sendto',
                  'setsockopt',
                  'shutdown']),
 'sendmsg': set(['accept4',
                 'bind',
                 'connect',
                 'getpeername',
                 'getsockname',
                 'getsockopt',
                 'listen',
                 'recvfrom',
                 'recvmmsg',
                 'recvmsg',
                 'sendmmsg',
                 'sendto',
                 'setsockopt',
                 'shutdown']),
 'sendto': set(['accept4',
                'acct',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'eventfd2',
                'flock',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_open',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'remap_file_pages',
                'setns',
                'shmat',
                'shmctl',
                'shmdt',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'uselib']),
 'set_mempolicy': set(['get_mempolicy', 'mbind']),
 'set_thread_area': set(['arch_prctl',
                         'capget',
                         'clone',
                         'fork',
```

```
                        'get_robust_list',
                        'getitimer',
                        'getpgid',
                        'getppid',
                        'getpriority',
                        'getrusage',
                        'getsid',
                        'ioperm',
                        'ioprio_get',
                        'ioprio_set',
                        'keyctl',
                        'kill',
                        'migrate_pages',
                        'move_pages',
                        'mq_timedreceive',
                        'mq_timedsend',
                        'msgrcv',
                        'perf_event_open',
                        'prctl',
                        'prlimit64',
                        'ptrace',
                        'rt_sigaction',
                        'rt_sigprocmask',
                        'rt_sigtimedwait',
                        'sched_getaffinity',
                        'sched_getattr',
                        'sched_getparam',
                        'sched_getscheduler',
                        'sched_rr_get_interval',
                        'sched_setaffinity',
                        'sched_setattr',
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setsid',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'umount',
                        'vfork']),
 'set_trip_temp': set(['get_curr_temp', 'get_trip_temp']),
 'setdomainname': set(['setns']),
 'setfsgid': set(['capget',
                  'capset',
                  'clone',
                  'epoll_create1',
                  'faccessat',
                  'fork',
                  'get_robust_list',
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
```



```
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsuid',
                  'setgid',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setregid',
                  'setresgid',
                  'setresuid',
                  'setreuid',
                  'setsid',
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setfsuid': set(['capget',
                  'capset',
                  'clone',
                  'epoll_create1',
                  'faccessat',
                  'fork',
                  'get_robust_list',
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
```

```
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsgid',
                  'setgid',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setregid',
                  'setresgid',
                  'setresuid',
                  'setreuid',
                  'setsid',
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setgid': set(['capget',
                'capset',
                'clone',
                'epoll_create1',
                'faccessat',
                'fork',
                'get_robust_list',
                'getgroups',
                'getgroups16',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getresgid',
                'getresgid16',
                'getresuid',
                'getresuid16',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
```



```
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setfsgid',
                'setfsuid',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setregid',
                'setresgid',
                'setresuid',
                'setreuid',
                'setsid',
                'setuid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'unshare',
                'vfork']),
 'setgroups16': set(['setgroups']),
 'sethostname': set(['setns']),
 'setitimer': set(['adjtimex',
                   'alarm',
                   'clock_adjtime',
                   'exit_group',
                   'getitimer',
                   'getrusage',
                   'ppoll',
                   'select',
                   'settimeofday',
                   'timer_create',
                   'wait4',
                   'waitid']),
 'setpgid': set(['capget',
                 'clone',
                 'exit_group',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
```

```
7650                 'migrate_pages',
7651                 'move_pages',
7652                 'mq_timedreceive',
7653                 'mq_timedsend',
7654                 'msgrcv',
7655                 'perf_event_open',
7656                 'prctl',
7657                 'prlimit64',
7658                 'ptrace',
7659                 'rt_sigaction',
7660                 'rt_sigprocmask',
7661                 'rt_sigtimedwait',
7662                 'sched_getaffinity',
7663                 'sched_getattr',
7664                 'sched_getparam',
7665                 'sched_getscheduler',
7666                 'sched_rr_get_interval',
7667                 'sched_setaffinity',
7668                 'sched_setattr',
7669                 'sched_setparam',
7670                 'sched_setscheduler',
7671                 'semtimedop',
7672                 'setitimer',
7673                 'setns',
7674                 'setpriority',
7675                 'setresuid',
7676                 'setreuid',
7677                 'setsid',
7678                 'setuid',
7679                 'sigaction',
7680                 'sigaltstack',
7681                 'signal',
7682                 'timer_create',
7683                 'umount',
7684                 'vfork']),
7685  'setpriority': set(['capget',
7686                      'clone',
7687                      'fork',
7688                      'get_robust_list',
7689                      'getitimer',
7690                      'getpgid',
7691                      'getppid',
7692                      'getpriority',
7693                      'getrusage',
7694                      'getsid',
7695                      'ioprio_get',
7696                      'ioprio_set',
7697                      'keyctl',
7698                      'kill',
7699                      'migrate_pages',
7700                      'move_pages',
7701                      'mq_timedreceive',
7702                      'mq_timedsend',
7703                      'msgrcv',
7704                      'perf_event_open',
7705                      'prctl',
7706                      'prlimit64',
7707                      'ptrace',
7708                      'rt_sigaction',
7709                      'rt_sigprocmask',
7710                      'rt_sigtimedwait',
7711                      'sched_getaffinity',
7712                      'sched_getattr',
7713                      'sched_getparam',
7714                      'sched_getscheduler',
7715                      'sched_rr_get_interval',
```



```
7716                      'sched_setaffinity',
7717                      'sched_setattr',
7718                      'sched_setparam',
7719                      'sched_setscheduler',
7720                      'semtimedop',
7721                      'setitimer',
7722                      'setns',
7723                      'setpgid',
7724                      'setresuid',
7725                      'setreuid',
7726                      'setsid',
7727                      'setuid',
7728                      'sigaction',
7729                      'sigaltstack',
7730                      'signal',
7731                      'umount',
7732                      'vfork']),
7733  'setregid': set(['capget',
7734                   'capset',
7735                   'clone',
7736                   'epoll_create1',
7737                   'faccessat',
7738                   'fork',
7739                   'get_robust_list',
7740                   'getgroups',
7741                   'getgroups16',
7742                   'getitimer',
7743                   'getpgid',
7744                   'getppid',
7745                   'getpriority',
7746                   'getresgid',
7747                   'getresgid16',
7748                   'getresuid',
7749                   'getresuid16',
7750                   'getrusage',
7751                   'getsid',
7752                   'ioprio_get',
7753                   'ioprio_set',
7754                   'keyctl',
7755                   'kill',
7756                   'migrate_pages',
7757                   'move_pages',
7758                   'mq_timedreceive',
7759                   'mq_timedsend',
7760                   'msgrcv',
7761                   'perf_event_open',
7762                   'prctl',
7763                   'prlimit64',
7764                   'ptrace',
7765                   'rt_sigaction',
7766                   'rt_sigprocmask',
7767                   'rt_sigtimedwait',
7768                   'sched_getaffinity',
7769                   'sched_getattr',
7770                   'sched_getparam',
7771                   'sched_getscheduler',
7772                   'sched_rr_get_interval',
7773                   'sched_setaffinity',
7774                   'sched_setattr',
7775                   'sched_setparam',
7776                   'sched_setscheduler',
7777                   'semtimedop',
7778                   'setfsgid',
7779                   'setfsuid',
7780                   'setgid',
7781                   'setitimer',
```

```
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setresgid',
                  'setresuid',
                  'setreuid',
                  'setsid',
                  'setuid',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setresgid': set(['capget',
                   'capset',
                   'clone',
                   'epoll_create1',
                   'faccessat',
                   'fork',
                   'get_robust_list',
                   'getgroups',
                   'getgroups16',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getresgid',
                   'getresgid16',
                   'getresuid',
                   'getresuid16',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setfsgid',
                   'setfsuid',
                   'setgid',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
```



```
                   'setregid',
                   'setresuid',
                   'setreuid',
                   'setsid',
                   'setuid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'unshare',
                   'vfork']),
 'setresuid': set(['capget',
                   'capset',
                   'clone',
                   'epoll_create1',
                   'faccessat',
                   'fork',
                   'get_robust_list',
                   'getgroups',
                   'getgroups16',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getresgid',
                   'getresgid16',
                   'getresuid',
                   'getresuid16',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setfsgid',
                   'setfsuid',
                   'setgid',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setregid',
                   'setresgid',
                   'setreuid',
```

```
                   'setsid',
                   'setuid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'unshare',
                   'vfork']),
 'setreuid': set(['capget',
                  'capset',
                  'clone',
                  'epoll_create1',
                  'faccessat',
                  'fork',
                  'get_robust_list',
                  'getgroups',
                  'getgroups16',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getresgid',
                  'getresgid16',
                  'getresuid',
                  'getresuid16',
                  'getrusage',
                  'getsid',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'migrate_pages',
                  'move_pages',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'perf_event_open',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setfsgid',
                  'setfsuid',
                  'setgid',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setregid',
                  'setresgid',
                  'setresuid',
                  'setsid',
                  'setuid',
                  'sigaction',
```



```
                  'sigaltstack',
                  'signal',
                  'umount',
                  'unshare',
                  'vfork']),
 'setrlimit': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrlimit',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'old_getrlimit',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'setsid': set(['exit_group', 'timer_create']),
 'setsockopt': set(['accept4']),
 'settimeofday': set(['adjtimex',
                      'alarm',
                      'clock_adjtime',
                      'getitimer',
                      'getrusage',
                      'ppoll',
                      'select',
                      'setitimer',
                      'wait4',
                      'waitid']),
 'setuid': set(['capget',
```

```
8046                 'capset',
8047                 'clone',
8048                 'epoll_create1',
8049                 'faccessat',
8050                 'fork',
8051                 'get_robust_list',
8052                 'getgroups',
8053                 'getgroups16',
8054                 'getitimer',
8055                 'getpgid',
8056                 'getppid',
8057                 'getpriority',
8058                 'getresgid',
8059                 'getresgid16',
8060                 'getresuid',
8061                 'getresuid16',
8062                 'getrusage',
8063                 'getsid',
8064                 'ioprio_get',
8065                 'ioprio_set',
8066                 'keyctl',
8067                 'kill',
8068                 'migrate_pages',
8069                 'move_pages',
8070                 'mq_timedreceive',
8071                 'mq_timedsend',
8072                 'msgrcv',
8073                 'perf_event_open',
8074                 'prctl',
8075                 'prlimit64',
8076                 'ptrace',
8077                 'rt_sigaction',
8078                 'rt_sigprocmask',
8079                 'rt_sigtimedwait',
8080                 'sched_getaffinity',
8081                 'sched_getattr',
8082                 'sched_getparam',
8083                 'sched_getscheduler',
8084                 'sched_rr_get_interval',
8085                 'sched_setaffinity',
8086                 'sched_setattr',
8087                 'sched_setparam',
8088                 'sched_setscheduler',
8089                 'semtimedop',
8090                 'setfsgid',
8091                 'setfsuid',
8092                 'setgid',
8093                 'setitimer',
8094                 'setns',
8095                 'setpgid',
8096                 'setpriority',
8097                 'setregid',
8098                 'setresgid',
8099                 'setresuid',
8100                 'setreuid',
8101                 'setsid',
8102                 'sigaction',
8103                 'sigaltstack',
8104                 'signal',
8105                 'umount',
8106                 'unshare',
8107                 'vfork']),
8108  'setxattr': set(['accept4',
8109                   'acct',
8110                   'dup',
8111                   'dup3',
```



```
8112                   'epoll_create1',
8113                   'epoll_ctl',
8114                   'eventfd2',
8115                   'flock',
8116                   'getcwd',
8117                   'lookup_dcookie',
8118                   'memfd_create',
8119                   'mmap_pgoff',
8120                   'mq_open',
8121                   'open',
8122                   'open_by_handle_at',
8123                   'openat',
8124                   'perf_event_open',
8125                   'pipe2',
8126                   'pivot_root',
8127                   'quotactl',
8128                   'remap_file_pages',
8129                   'setns',
8130                   'shmat',
8131                   'shmctl',
8132                   'shmdt',
8133                   'socket',
8134                   'socketpair',
8135                   'swapoff',
8136                   'swapon',
8137                   'unshare',
8138                   'uselib']),
8139  'shmat': set(['accept4',
8140                'acct',
8141                'capget',
8142                'clone',
8143                'dup',
8144                'dup3',
8145                'epoll_create1',
8146                'epoll_ctl',
8147                'eventfd2',
8148                'flock',
8149                'fork',
8150                'get_robust_list',
8151                'getcwd',
8152                'getitimer',
8153                'getpgid',
8154                'getppid',
8155                'getpriority',
8156                'getrusage',
8157                'getsid',
8158                'ioprio_get',
8159                'ioprio_set',
8160                'keyctl',
8161                'kill',
8162                'lookup_dcookie',
8163                'memfd_create',
8164                'migrate_pages',
8165                'mmap_pgoff',
8166                'move_pages',
8167                'mq_open',
8168                'mq_timedreceive',
8169                'mq_timedsend',
8170                'msgrcv',
8171                'open',
8172                'open_by_handle_at',
8173                'openat',
8174                'perf_event_open',
8175                'pipe2',
8176                'pivot_root',
8177                'prctl',
```

```
8178                'prlimit64',
8179                'ptrace',
8180                'quotactl',
8181                'remap_file_pages',
8182                'rt_sigaction',
8183                'rt_sigprocmask',
8184                'rt_sigtimedwait',
8185                'sched_getaffinity',
8186                'sched_getattr',
8187                'sched_getparam',
8188                'sched_getscheduler',
8189                'sched_rr_get_interval',
8190                'sched_setaffinity',
8191                'sched_setattr',
8192                'sched_setparam',
8193                'sched_setscheduler',
8194                'semtimedop',
8195                'setitimer',
8196                'setns',
8197                'setpgid',
8198                'setpriority',
8199                'setsid',
8200                'shmctl',
8201                'shmdt',
8202                'sigaction',
8203                'sigaltstack',
8204                'signal',
8205                'socket',
8206                'socketpair',
8207                'swapoff',
8208                'swapon',
8209                'umount',
8210                'unshare',
8211                'uselib',
8212                'vfork']),
8213  'shmctl': set(['accept4',
8214                 'acct',
8215                 'capget',
8216                 'clone',
8217                 'dup',
8218                 'dup3',
8219                 'epoll_create1',
8220                 'epoll_ctl',
8221                 'eventfd2',
8222                 'flock',
8223                 'fork',
8224                 'get_robust_list',
8225                 'getitimer',
8226                 'getpgid',
8227                 'getppid',
8228                 'getpriority',
8229                 'getrusage',
8230                 'getsid',
8231                 'ioperm',
8232                 'ioprio_get',
8233                 'ioprio_set',
8234                 'keyctl',
8235                 'kill',
8236                 'memfd_create',
8237                 'migrate_pages',
8238                 'mmap_pgoff',
8239                 'move_pages',
8240                 'mq_open',
8241                 'mq_timedreceive',
8242                 'mq_timedsend',
8243                 'mq_unlink',
```



```
8244                'msgctl',
8245                'msgget',
8246                'msgrcv',
8247                'msgsnd',
8248                'open',
8249                'open_by_handle_at',
8250                'openat',
8251                'perf_event_open',
8252                'pipe2',
8253                'prctl',
8254                'prlimit64',
8255                'ptrace',
8256                'remap_file_pages',
8257                'rt_sigaction',
8258                'rt_sigprocmask',
8259                'rt_sigtimedwait',
8260                'sched_getaffinity',
8261                'sched_getattr',
8262                'sched_getparam',
8263                'sched_getscheduler',
8264                'sched_rr_get_interval',
8265                'sched_setaffinity',
8266                'sched_setattr',
8267                'sched_setparam',
8268                'sched_setscheduler',
8269                'semctl',
8270                'semget',
8271                'semtimedop',
8272                'setitimer',
8273                'setns',
8274                'setpgid',
8275                'setpriority',
8276                'setsid',
8277                'shmat',
8278                'shmdt',
8279                'shmget',
8280                'sigaction',
8281                'sigaltstack',
8282                'signal',
8283                'socket',
8284                'socketpair',
8285                'swapoff',
8286                'swapon',
8287                'umount',
8288                'uselib',
8289                'vfork']),
8290  'shmdt': set(['brk',
8291                'get_mempolicy',
8292                'madvise',
8293                'mincore',
8294                'mlockall',
8295                'mprotect',
8296                'mremap',
8297                'munlock',
8298                'munlockall',
8299                'pkey_mprotect',
8300                'prctl',
8301                'remap_file_pages']),
8302  'sigaction': set(['capget',
8303                    'clone',
8304                    'fork',
8305                    'get_robust_list',
8306                    'getitimer',
8307                    'getpgid',
8308                    'getppid',
8309                    'getpriority',
```

```
                    'getrusage',
                    'getsid',
                    'ioperm',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'signalfd4': set(['accept4',
                    'acct',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'epoll_wait',
                    'eventfd2',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
```



```
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setns',
                    'setsockopt',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'shutdown',
                    'socket',
                    'socketpair',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'splice': set(['accept4',
                'acct',
                'bind',
```

```
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
```



```
                'sendfile64',
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
    'stat': set(['fstat', 'lstat', 'newfstat']),
    'statfs': set(['fstatfs', 'fstatfs64', 'statfs64', 'ustat']),
    'statfs64': set(['fstatfs', 'fstatfs64', 'statfs', 'ustat']),
    'swapoff': set(['accept4',
                    'acct',
                    'capget',
                    'clone',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'flock',
                    'fork',
                    'get_robust_list',
                    'getitimer',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'memfd_create',
                    'migrate_pages',
                    'mmap_pgoff',
                    'move_pages',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'remap_file_pages',
                    'rt_sigaction',
                    'rt_sigprocmask',
```

```
8574                 'rt_sigtimedwait',
8575                 'sched_getaffinity',
8576                 'sched_getattr',
8577                 'sched_getparam',
8578                 'sched_getscheduler',
8579                 'sched_rr_get_interval',
8580                 'sched_setaffinity',
8581                 'sched_setattr',
8582                 'sched_setparam',
8583                 'sched_setscheduler',
8584                 'semtimedop',
8585                 'setitimer',
8586                 'setns',
8587                 'setpgid',
8588                 'setpriority',
8589                 'setsid',
8590                 'shmat',
8591                 'shmctl',
8592                 'shmdt',
8593                 'sigaction',
8594                 'sigaltstack',
8595                 'signal',
8596                 'socket',
8597                 'socketpair',
8598                 'swapon',
8599                 'umount',
8600                 'uselib',
8601                 'vfork']),
8602  'swapon': set(['faccessat',
8603                 'fchmod',
8604                 'fchmodat',
8605                 'fchown',
8606                 'fchownat',
8607                 'ftruncate',
8608                 'inotify_add_watch',
8609                 'ioctl',
8610                 'linkat',
8611                 'memfd_create',
8612                 'mq_getsetattr',
8613                 'mq_notify',
8614                 'mq_timedreceive',
8615                 'mq_timedsend',
8616                 'mq_unlink',
8617                 'readlinkat',
8618                 'sendfile',
8619                 'sendfile64',
8620                 'swapoff',
8621                 'unlink',
8622                 'unlinkat',
8623                 'uselib']),
8624  'symlinkat': set(['acct',
8625                    'mq_open',
8626                    'mq_unlink',
8627                    'open',
8628                    'openat',
8629                    'quotactl',
8630                    'renameat2',
8631                    'rmdir',
8632                    'swapoff',
8633                    'swapon',
8634                    'sysfs',
8635                    'unlink',
8636                    'unlinkat',
8637                    'uselib']),
8638  'sync_file_range': set(['accept4',
8639                          'bind',
```



```
8640                          'bpf',
8641                          'connect',
8642                          'copy_file_range',
8643                          'epoll_ctl',
8644                          'epoll_wait',
8645                          'fallocate',
8646                          'fchdir',
8647                          'fchmod',
8648                          'fchown',
8649                          'fcntl',
8650                          'fcntl64',
8651                          'fdatasync',
8652                          'fgetxattr',
8653                          'flistxattr',
8654                          'flock',
8655                          'fremovexattr',
8656                          'fsetxattr',
8657                          'fstatfs',
8658                          'fstatfs64',
8659                          'fsync',
8660                          'ftruncate',
8661                          'futimesat',
8662                          'getdents',
8663                          'getdents64',
8664                          'getpeername',
8665                          'getsockname',
8666                          'getsockopt',
8667                          'inotify_add_watch',
8668                          'inotify_rm_watch',
8669                          'ioctl',
8670                          'listen',
8671                          'llseek',
8672                          'lseek',
8673                          'mq_getsetattr',
8674                          'mq_notify',
8675                          'mq_timedreceive',
8676                          'mq_timedsend',
8677                          'old_readdir',
8678                          'perf_event_open',
8679                          'pread64',
8680                          'preadv',
8681                          'preadv2',
8682                          'preadv64',
8683                          'preadv64v2',
8684                          'pwrite64',
8685                          'pwritev',
8686                          'pwritev2',
8687                          'pwritev64',
8688                          'pwritev64v2',
8689                          'read',
8690                          'readahead',
8691                          'readv',
8692                          'recvfrom',
8693                          'sendfile',
8694                          'sendfile64',
8695                          'sendto',
8696                          'setsockopt',
8697                          'shutdown',
8698                          'signalfd4',
8699                          'splice',
8700                          'syncfs',
8701                          'tee',
8702                          'utime',
8703                          'utimensat',
8704                          'vmsplice',
8705                          'write',
```

```
                        'writev']),
 'syncfs': set(['accept4',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'epoll_ctl',
                'epoll_wait',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'perf_event_open',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'tee',
                'utime',
```



```
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'sysfs': set(['acct',
               'mq_open',
               'mq_unlink',
               'open',
               'openat',
               'quotactl',
               'renameat2',
               'rmdir',
               'swapoff',
               'swapon',
               'symlinkat',
               'unlink',
               'unlinkat',
               'uselib']),
 'sysinfo': set(['capget',
                 'clock_nanosleep',
                 'clone',
                 'epoll_wait',
                 'faccessat',
                 'fchmod',
                 'fchmodat',
                 'fchown',
                 'fchownat',
                 'fork',
                 'fstat',
                 'ftruncate',
                 'futex',
                 'futimesat',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'inotify_add_watch',
                 'io_getevents',
                 'ioctl',
                 'ioperm',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'linkat',
                 'memfd_create',
                 'migrate_pages',
                 'move_pages',
                 'mq_getsetattr',
                 'mq_notify',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'mq_unlink',
                 'msgrcv',
                 'nanosleep',
                 'newfstat',
                 'perf_event_open',
                 'poll',
                 'ppoll',
                 'prctl',
                 'prlimit64',
                 'pselect6',
                 'ptrace',
```

```
                'readlinkat',
                'recvmmsg',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'select',
                'semtimedop',
                'sendfile',
                'sendfile64',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'settimeofday',
                'sigaction',
                'sigaltstack',
                'signal',
                'stime',
                'swapoff',
                'swapon',
                'timer_gettime',
                'timer_settime',
                'timerfd_gettime',
                'timerfd_settime',
                'umount',
                'unlink',
                'unlinkat',
                'uselib',
                'utime',
                'vfork']),
 'syslog': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioperm',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
```



```
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'tee': set(['accept4',
             'acct',
             'bind',
             'bpf',
             'connect',
             'copy_file_range',
             'dup',
             'dup3',
             'epoll_create1',
             'epoll_ctl',
             'epoll_wait',
             'eventfd2',
             'fallocate',
             'fchdir',
             'fchmod',
             'fchown',
             'fcntl',
             'fcntl64',
             'fdatasync',
             'fgetxattr',
             'flistxattr',
             'flock',
             'fremovexattr',
             'fsetxattr',
             'fstatfs',
             'fstatfs64',
             'fsync',
             'ftruncate',
             'futimesat',
             'getdents',
             'getdents64',
             'getpeername',
             'getsockname',
             'getsockopt',
             'inotify_add_watch',
             'inotify_rm_watch',
             'ioctl',
             'listen',
             'llseek',
             'lseek',
             'memfd_create',
             'mmap_pgoff',
             'mq_getsetattr',
             'mq_notify',
             'mq_open',
```

```
8970              'mq_timedreceive',
8971              'mq_timedsend',
8972              'old_readdir',
8973              'open',
8974              'open_by_handle_at',
8975              'openat',
8976              'perf_event_open',
8977              'pipe2',
8978              'pread64',
8979              'preadv',
8980              'preadv2',
8981              'preadv64',
8982              'preadv64v2',
8983              'pwrite64',
8984              'pwritev',
8985              'pwritev2',
8986              'pwritev64',
8987              'pwritev64v2',
8988              'read',
8989              'readahead',
8990              'readv',
8991              'recvfrom',
8992              'remap_file_pages',
8993              'sendfile',
8994              'sendfile64',
8995              'sendto',
8996              'setns',
8997              'setsockopt',
8998              'shmat',
8999              'shmctl',
9000              'shmdt',
9001              'shutdown',
9002              'signalfd4',
9003              'socket',
9004              'socketpair',
9005              'splice',
9006              'swapoff',
9007              'swapon',
9008              'sync_file_range',
9009              'syncfs',
9010              'uselib',
9011              'utime',
9012              'utimensat',
9013              'vmsplice',
9014              'write',
9015              'writev']),
9016  'tgkill': set(['capget',
9017                 'clone',
9018                 'fork',
9019                 'get_robust_list',
9020                 'getitimer',
9021                 'getpgid',
9022                 'getppid',
9023                 'getpriority',
9024                 'getrusage',
9025                 'getsid',
9026                 'ioprio_get',
9027                 'ioprio_set',
9028                 'keyctl',
9029                 'kill',
9030                 'migrate_pages',
9031                 'move_pages',
9032                 'mq_timedreceive',
9033                 'mq_timedsend',
9034                 'msgrcv',
9035                 'perf_event_open',
```



```
9036                 'prctl',
9037                 'prlimit64',
9038                 'ptrace',
9039                 'rt_sigaction',
9040                 'rt_sigprocmask',
9041                 'rt_sigtimedwait',
9042                 'sched_getaffinity',
9043                 'sched_getattr',
9044                 'sched_getparam',
9045                 'sched_getscheduler',
9046                 'sched_rr_get_interval',
9047                 'sched_setaffinity',
9048                 'sched_setattr',
9049                 'sched_setparam',
9050                 'sched_setscheduler',
9051                 'semtimedop',
9052                 'setitimer',
9053                 'setns',
9054                 'setpgid',
9055                 'setpriority',
9056                 'setsid',
9057                 'sigaction',
9058                 'sigaltstack',
9059                 'signal',
9060                 'umount',
9061                 'vfork']),
9062  'timer_create': set(['clock_adjtime',
9063                       'clock_getres',
9064                       'clock_gettime',
9065                       'clock_nanosleep',
9066                       'clock_settime',
9067                       'timer_delete',
9068                       'timer_getoverrun',
9069                       'timer_gettime',
9070                       'timer_settime']),
9071  'timer_delete': set(['capget',
9072                       'clock_adjtime',
9073                       'clock_getres',
9074                       'clock_gettime',
9075                       'clock_nanosleep',
9076                       'clock_settime',
9077                       'clone',
9078                       'fork',
9079                       'get_robust_list',
9080                       'getitimer',
9081                       'getpgid',
9082                       'getppid',
9083                       'getpriority',
9084                       'getrusage',
9085                       'getsid',
9086                       'ioprio_get',
9087                       'ioprio_set',
9088                       'keyctl',
9089                       'kill',
9090                       'migrate_pages',
9091                       'move_pages',
9092                       'mq_timedreceive',
9093                       'mq_timedsend',
9094                       'msgrcv',
9095                       'perf_event_open',
9096                       'prctl',
9097                       'prlimit64',
9098                       'ptrace',
9099                       'rt_sigaction',
9100                       'rt_sigprocmask',
9101                       'rt_sigtimedwait',
```

```
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setsid',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'timer_create',
                      'timer_getoverrun',
                      'timer_gettime',
                      'timer_settime',
                      'umount',
                      'vfork']),
 'timer_getoverrun': set(['capget',
                          'clone',
                          'fork',
                          'get_robust_list',
                          'getitimer',
                          'getpgid',
                          'getppid',
                          'getpriority',
                          'getrusage',
                          'getsid',
                          'ioprio_get',
                          'ioprio_set',
                          'keyctl',
                          'kill',
                          'migrate_pages',
                          'move_pages',
                          'mq_timedreceive',
                          'mq_timedsend',
                          'msgrcv',
                          'perf_event_open',
                          'prctl',
                          'prlimit64',
                          'ptrace',
                          'rt_sigaction',
                          'rt_sigprocmask',
                          'rt_sigtimedwait',
                          'sched_getaffinity',
                          'sched_getattr',
                          'sched_getparam',
                          'sched_getscheduler',
                          'sched_rr_get_interval',
                          'sched_setaffinity',
                          'sched_setattr',
                          'sched_setparam',
                          'sched_setscheduler',
                          'semtimedop',
                          'setitimer',
                          'setns',
                          'setpgid',
                          'setpriority',
                          'setsid',
                          'sigaction',
```



```
                          'sigaltstack',
                          'signal',
                          'timer_create',
                          'timer_delete',
                          'timer_gettime',
                          'timer_settime',
                          'umount',
                          'vfork']),
 'timer_gettime': set(['clock_adjtime',
                       'clock_getres',
                       'clock_gettime',
                       'clock_nanosleep',
                       'clock_settime',
                       'timer_create',
                       'timer_delete',
                       'timer_settime']),
 'timer_settime': set(['clock_adjtime',
                       'clock_getres',
                       'clock_gettime',
                       'clock_nanosleep',
                       'clock_settime',
                       'timer_create',
                       'timer_delete',
                       'timer_gettime']),
 'timerfd_create': set(['timerfd_gettime', 'timerfd_settime']),
 'timerfd_gettime': set(['timerfd_create', 'timerfd_settime']),
 'timerfd_settime': set(['timerfd_create', 'timerfd_gettime']),
 'tkill': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'migrate_pages',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'msgrcv',
               'perf_event_open',
               'prctl',
               'prlimit64',
               'ptrace',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
               'semtimedop',
               'setitimer',
               'setns',
               'setpgid',
```

```
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'umount': set(['accept4',
                'acct',
                'capget',
                'clone',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'eventfd2',
                'flock',
                'fork',
                'get_robust_list',
                'getcwd',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'lookup_dcookie',
                'memfd_create',
                'migrate_pages',
                'mmap_pgoff',
                'move_pages',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'msgrcv',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'prctl',
                'prlimit64',
                'ptrace',
                'quotactl',
                'remap_file_pages',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
```



```
                'setpgid',
                'setpriority',
                'setresuid',
                'setreuid',
                'setsid',
                'setuid',
                'shmat',
                'shmctl',
                'shmdt',
                'sigaction',
                'sigaltstack',
                'signal',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'syncfs',
                'unshare',
                'uselib',
                'ustat',
                'vfork']),
 'uname': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'migrate_pages',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'msgrcv',
               'perf_event_open',
               'personality',
               'prctl',
               'prlimit64',
               'ptrace',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
               'semtimedop',
               'setitimer',
               'setns',
               'setpgid',
               'setpriority',
               'setsid',
               'sigaction',
               'sigaltstack',
               'signal',
```

```
                'umount',
                'vfork']),
 'unlink': set(['accept4',
                'acct',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'eventfd2',
                'flock',
                'ftruncate',
                'getcwd',
                'linkat',
                'lookup_dcookie',
                'memfd_create',
                'mkdirat',
                'mknodat',
                'mmap_pgoff',
                'mq_open',
                'mq_unlink',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pivot_root',
                'quotactl',
                'remap_file_pages',
                'renameat2',
                'rmdir',
                'setns',
                'shmat',
                'shmctl',
                'shmdt',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'symlinkat',
                'unlinkat',
                'unshare',
                'uselib']),
 'unlinkat': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'flock',
                  'ftruncate',
                  'getcwd',
                  'linkat',
                  'lookup_dcookie',
                  'memfd_create',
                  'mkdirat',
                  'mknodat',
                  'mmap_pgoff',
                  'mq_open',
                  'mq_unlink',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'pivot_root',
```



```
                  'quotactl',
                  'remap_file_pages',
                  'renameat2',
                  'rmdir',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'symlinkat',
                  'unlink',
                  'unshare',
                  'uselib']),
 'uselib': set(['acct',
                'getcwd',
                'mq_open',
                'mq_unlink',
                'pivot_root',
                'quotactl',
                'swapon',
                'syncfs',
                'umount',
                'ustat']),
 'ustat': set(['quotactl', 'swapon', 'syncfs', 'umount']),
 'utime': set(['clock_nanosleep',
               'epoll_wait',
               'faccessat',
               'fchmod',
               'fchmodat',
               'fchown',
               'fchownat',
               'fstat',
               'ftruncate',
               'futex',
               'futimesat',
               'inotify_add_watch',
               'io_getevents',
               'ioctl',
               'linkat',
               'memfd_create',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'mq_unlink',
               'nanosleep',
               'newfstat',
               'poll',
               'ppoll',
               'pselect6',
               'readlinkat',
               'recvmmsg',
               'rt_sigtimedwait',
               'sched_rr_get_interval',
               'select',
               'semtimedop',
               'sendfile',
               'sendfile64',
               'settimeofday',
               'stime',
               'swapoff',
               'swapon',
               'timer_gettime',
```

```
9498                'timer_settime',
9499                'timerfd_gettime',
9500                'timerfd_settime',
9501                'unlink',
9502                'unlinkat',
9503                'uselib']),
9504  'utimensat': set(['clock_nanosleep',
9505                    'epoll_wait',
9506                    'faccessat',
9507                    'fchmod',
9508                    'fchmodat',
9509                    'fchown',
9510                    'fchownat',
9511                    'fstat',
9512                    'ftruncate',
9513                    'futex',
9514                    'futimesat',
9515                    'inotify_add_watch',
9516                    'io_getevents',
9517                    'ioctl',
9518                    'linkat',
9519                    'memfd_create',
9520                    'mq_getsetattr',
9521                    'mq_notify',
9522                    'mq_timedreceive',
9523                    'mq_timedsend',
9524                    'mq_unlink',
9525                    'nanosleep',
9526                    'newfstat',
9527                    'poll',
9528                    'ppoll',
9529                    'pselect6',
9530                    'readlinkat',
9531                    'recvmmsg',
9532                    'rt_sigtimedwait',
9533                    'sched_rr_get_interval',
9534                    'select',
9535                    'semtimedop',
9536                    'sendfile',
9537                    'sendfile64',
9538                    'settimeofday',
9539                    'stime',
9540                    'swapoff',
9541                    'swapon',
9542                    'timer_gettime',
9543                    'timer_settime',
9544                    'timerfd_gettime',
9545                    'timerfd_settime',
9546                    'unlink',
9547                    'unlinkat',
9548                    'uselib',
9549                    'utime']),
9550  'vmsplice': set(['accept4',
9551                   'acct',
9552                   'bind',
9553                   'bpf',
9554                   'connect',
9555                   'copy_file_range',
9556                   'dup',
9557                   'dup3',
9558                   'epoll_create1',
9559                   'epoll_ctl',
9560                   'epoll_wait',
9561                   'eventfd2',
9562                   'fallocate',
9563                   'fchdir',
```



```
9564                   'fchmod',
9565                   'fchown',
9566                   'fcntl',
9567                   'fcntl64',
9568                   'fdatasync',
9569                   'fgetxattr',
9570                   'flistxattr',
9571                   'flock',
9572                   'fremovexattr',
9573                   'fsetxattr',
9574                   'fstatfs',
9575                   'fstatfs64',
9576                   'fsync',
9577                   'ftruncate',
9578                   'futimesat',
9579                   'getdents',
9580                   'getdents64',
9581                   'getpeername',
9582                   'getsockname',
9583                   'getsockopt',
9584                   'inotify_add_watch',
9585                   'inotify_rm_watch',
9586                   'ioctl',
9587                   'listen',
9588                   'llseek',
9589                   'lseek',
9590                   'memfd_create',
9591                   'mmap_pgoff',
9592                   'mq_getsetattr',
9593                   'mq_notify',
9594                   'mq_open',
9595                   'mq_timedreceive',
9596                   'mq_timedsend',
9597                   'old_readdir',
9598                   'open',
9599                   'open_by_handle_at',
9600                   'openat',
9601                   'perf_event_open',
9602                   'pipe2',
9603                   'pread64',
9604                   'preadv',
9605                   'preadv2',
9606                   'preadv64',
9607                   'preadv64v2',
9608                   'pwrite64',
9609                   'pwritev',
9610                   'pwritev2',
9611                   'pwritev64',
9612                   'pwritev64v2',
9613                   'read',
9614                   'readahead',
9615                   'readv',
9616                   'recvfrom',
9617                   'remap_file_pages',
9618                   'sendfile',
9619                   'sendfile64',
9620                   'sendto',
9621                   'setns',
9622                   'setsockopt',
9623                   'shmat',
9624                   'shmctl',
9625                   'shmdt',
9626                   'shutdown',
9627                   'signalfd4',
9628                   'socket',
9629                   'socketpair',
```

```
                  'splice',
                  'swapoff',
                  'swapon',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'uselib',
                  'utime',
                  'utimensat',
                  'write',
                  'writev']),
 'write': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
```



```
               'sendfile',
               'sendfile64',
               'sendto',
               'setsockopt',
               'shutdown',
               'signalfd4',
               'splice',
               'sync_file_range',
               'syncfs',
               'tee',
               'utime',
               'utimensat',
               'vmsplice',
               'writev'])}
```

```
**********************************************************
 1337256 Fri Dec 21 15:00:32 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/with_structs/i
mplicit_dependencies_verbose
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
{'accept4': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
             {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
             {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
             {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
             {'call': 'readahead', 'reason': set([('fd', 'file')])},
             {'call': 'getdents', 'reason': set([('fd', 'file')])},
             {'call': 'writev', 'reason': set([('fd', 'file')])},
             {'call': 'preadv64', 'reason': set([('fd', 'file')])},
             {'call': 'fchmod', 'reason': set([('fd', 'file')])},
             {'call': 'pread64', 'reason': set([('fd', 'file')])},
             {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
             {'call': 'read', 'reason': set([('fd', 'file')])},
             {'call': 'fchown', 'reason': set([('fd', 'file')])},
             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
             {'call': 'utime', 'reason': set([('fd', 'file')])},
             {'call': 'fsync', 'reason': set([('fd', 'file')])},
             {'call': 'bpf', 'reason': set([('fd', 'file')])},
             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
             {'call': 'sendto', 'reason': set([('fd', 'file')])},
             {'call': 'tee', 'reason': set([('fd', 'file')])},
             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
             {'call': 'lseek', 'reason': set([('fd', 'file')])},
             {'call': 'connect', 'reason': set([('fd', 'file')])},
             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
             {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
             {'call': 'flock', 'reason': set([('fd', 'file')])},
             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
             {'call': 'accept4', 'reason': set([('fd', 'file')])},
             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
             {'call': 'splice', 'reason': set([('fd', 'file')])},
             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
             {'call': 'preadv', 'reason': set([('fd', 'file')])},
             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
             {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
             {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
             {'call': 'listen', 'reason': set([('fd', 'file')])},
             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
```



```
             {'call': 'llseek', 'reason': set([('fd', 'file')])},
             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
             {'call': 'readv', 'reason': set([('fd', 'file')])},
             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
             {'call': 'write', 'reason': set([('fd', 'file')])},
             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
             {'call': 'bind', 'reason': set([('fd', 'file')])},
             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
 'acct': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
          {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
          {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
          {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
          {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
          {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
          {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
          {'call': 'flock', 'reason': set([('file', 'f_mode')])},
          {'call': 'openat', 'reason': set([('file', 'f_mode')])},
          {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
          {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
          {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
          {'call': 'socket', 'reason': set([('file', 'f_mode')])},
          {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
          {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
          {'call': 'acct', 'reason': set([('file', 'f_mode')])},
          {'call': 'open', 'reason': set([('file', 'f_mode')])},
          {'call': 'dup', 'reason': set([('file', 'f_mode')])},
          {'call': 'setns', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
          {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
          {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
          {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
          {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])}],
 'alarm': [{'call': 'waitid',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'settimeofday',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'adjtimex',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'getitimer',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'select',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'wait4',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'getrusage',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'setitimer',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'clock_adjtime',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'alarm',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'ppoll',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])}],
 'bind': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
          {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
          {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
          {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
          {'call': 'readahead', 'reason': set([('fd', 'file')])},
          {'call': 'getdents', 'reason': set([('fd', 'file')])},
          {'call': 'writev', 'reason': set([('fd', 'file')])},
```

```
126          {'call': 'preadv64', 'reason': set([('fd', 'file')])},
127          {'call': 'fchmod', 'reason': set([('fd', 'file')])},
128          {'call': 'pread64', 'reason': set([('fd', 'file')])},
129          {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
130          {'call': 'read', 'reason': set([('fd', 'file')])},
131          {'call': 'fchown', 'reason': set([('fd', 'file')])},
132          {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
133          {'call': 'utime', 'reason': set([('fd', 'file')])},
134          {'call': 'fsync', 'reason': set([('fd', 'file')])},
135          {'call': 'bpf', 'reason': set([('fd', 'file')])},
136          {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
137          {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
138          {'call': 'sendto', 'reason': set([('fd', 'file')])},
139          {'call': 'tee', 'reason': set([('fd', 'file')])},
140          {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
141          {'call': 'lseek', 'reason': set([('fd', 'file')])},
142          {'call': 'connect', 'reason': set([('fd', 'file')])},
143          {'call': 'getsockname', 'reason': set([('fd', 'file')])},
144          {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
145          {'call': 'flock', 'reason': set([('fd', 'file')])},
146          {'call': 'pwritev', 'reason': set([('fd', 'file')])},
147          {'call': 'fchdir', 'reason': set([('fd', 'file')])},
148          {'call': 'accept4', 'reason': set([('fd', 'file')])},
149          {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
150          {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
151          {'call': 'utimensat', 'reason': set([('fd', 'file')])},
152          {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
153          {'call': 'preadv2', 'reason': set([('fd', 'file')])},
154          {'call': 'splice', 'reason': set([('fd', 'file')])},
155          {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
156          {'call': 'preadv', 'reason': set([('fd', 'file')])},
157          {'call': 'getpeername', 'reason': set([('fd', 'file')])},
158          {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
159          {'call': 'fcntl', 'reason': set([('fd', 'file')])},
160          {'call': 'ioctl', 'reason': set([('fd', 'file')])},
161          {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
162          {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
163          {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
164          {'call': 'futimesat', 'reason': set([('fd', 'file')])},
165          {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
166          {'call': 'shutdown', 'reason': set([('fd', 'file')])},
167          {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
168          {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
169          {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
170          {'call': 'getdents64', 'reason': set([('fd', 'file')])},
171          {'call': 'listen', 'reason': set([('fd', 'file')])},
172          {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
173          {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
174          {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
175          {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
176          {'call': 'fallocate', 'reason': set([('fd', 'file')])},
177          {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
178          {'call': 'llseek', 'reason': set([('fd', 'file')])},
179          {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
180          {'call': 'readv', 'reason': set([('fd', 'file')])},
181          {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
182          {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
183          {'call': 'write', 'reason': set([('fd', 'file')])},
184          {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
185          {'call': 'sendfile', 'reason': set([('fd', 'file')])},
186          {'call': 'bind', 'reason': set([('fd', 'file')])},
187          {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
188          {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
189  'bpf': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
190          {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
191          {'call': 'rt_sigtimedwait',
```



```
192           'reason': set([('mm_segment_t', 'seg')])},
193          {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
194          {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
195          {'call': 'eventfd2',
196           'reason': set([('file', 'f_op'), ('file', 'private_data')])},
197          {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
198          {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
199          {'call': 'swapoff',
200           'reason': set([('file', 'f_op'), ('file', 'private_data')])},
201          {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
202          {'call': 'readahead', 'reason': set([('fd', 'file')])},
203          {'call': 'getdents', 'reason': set([('fd', 'file')])},
204          {'call': 'sched_getaffinity',
205           'reason': set([('mm_segment_t', 'seg')])},
206          {'call': 'writev', 'reason': set([('fd', 'file')])},
207          {'call': 'preadv64', 'reason': set([('fd', 'file')])},
208          {'call': 'sched_setparam', 'reason': set([('mm_segment_t', 'seg')])},
209          {'call': 'fchmod', 'reason': set([('fd', 'file')])},
210          {'call': 'pread64', 'reason': set([('fd', 'file')])},
211          {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
212          {'call': 'memfd_create',
213           'reason': set([('file', 'f_op'), ('file', 'private_data')])},
214          {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
215          {'call': 'remap_file_pages',
216           'reason': set([('file', 'f_op'), ('file', 'private_data')])},
217          {'call': 'dup3',
218           'reason': set([('file', 'f_op'), ('file', 'private_data')])},
219          {'call': 'read', 'reason': set([('fd', 'file')])},
220          {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
221          {'call': 'fchown', 'reason': set([('fd', 'file')])},
222          {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
223          {'call': 'mq_timedreceive',
224           'reason': set([('fd', 'file'), ('mm_segment_t', 'seg')])},
225          {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
226          {'call': 'utime', 'reason': set([('fd', 'file')])},
227          {'call': 'sched_setaffinity',
228           'reason': set([('mm_segment_t', 'seg')])},
229          {'call': 'fsync', 'reason': set([('fd', 'file')])},
230          {'call': 'bpf',
231           'reason': set([('bpf_attr', 'bpf_fd'),
232                          ('bpf_attr', 'flags'),
233                          ('bpf_attr', 'insn_cnt'),
234                          ('bpf_attr', 'kern_version'),
235                          ('bpf_attr', 'prog_flags'),
236                          ('bpf_map', 'key_size'),
237                          ('bpf_map', 'map_type'),
238                          ('bpf_prog', 'aux'),
239                          ('fd', 'file')])},
240          {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
241          {'call': 'semtimedop', 'reason': set([('mm_segment_t', 'seg')])},
242          {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
243          {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
244          {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
245          {'call': 'sendto', 'reason': set([('fd', 'file')])},
246          {'call': 'sched_rr_get_interval',
247           'reason': set([('mm_segment_t', 'seg')])},
248          {'call': 'epoll_create1',
249           'reason': set([('file', 'f_op'), ('file', 'private_data')])},
250          {'call': 'tee', 'reason': set([('fd', 'file')])},
251          {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
252          {'call': 'lseek', 'reason': set([('fd', 'file')])},
253          {'call': 'connect', 'reason': set([('fd', 'file')])},
254          {'call': 'getsockname', 'reason': set([('fd', 'file')])},
255          {'call': 'epoll_ctl',
256           'reason': set([('fd', 'file'),
257                          ('file', 'f_op'),
```

```
                        ('file', 'private_data')])},
        {'call': 'flock',
         'reason': set([('fd', 'file'),
                        ('file', 'f_op'),
                        ('file', 'private_data')])},
        {'call': 'pwritev', 'reason': set([('fd', 'file')])},
        {'call': 'fchdir', 'reason': set([('fd', 'file')])},
        {'call': 'openat',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'uselib',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'rt_sigprocmask', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'accept4',
         'reason': set([('fd', 'file'),
                        ('file', 'f_op'),
                        ('file', 'private_data')])},
        {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'sched_setattr', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
        {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
        {'call': 'socketpair',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'utimensat', 'reason': set([('fd', 'file')])},
        {'call': 'migrate_pages', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
        {'call': 'preadv2', 'reason': set([('fd', 'file')])},
        {'call': 'splice', 'reason': set([('fd', 'file')])},
        {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
        {'call': 'preadv', 'reason': set([('fd', 'file')])},
        {'call': 'getpeername', 'reason': set([('fd', 'file')])},
        {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'shmat',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
        {'call': 'socket',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'pipe2',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'fcntl', 'reason': set([('fd', 'file')])},
        {'call': 'ioctl', 'reason': set([('fd', 'file')])},
        {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
        {'call': 'perf_event_open',
         'reason': set([('bpf_prog', 'aux'),
                        ('fd', 'file'),
                        ('file', 'f_op'),
                        ('file', 'private_data'),
                        ('mm_segment_t', 'seg')])},
        {'call': 'shmdt',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
        {'call': 'rt_sigaction', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'futimesat', 'reason': set([('fd', 'file')])},
        {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
        {'call': 'shutdown', 'reason': set([('fd', 'file')])},
        {'call': 'acct',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'open',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
        {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
```



```
        {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
        {'call': 'dup',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
        {'call': 'setns',
         'reason': set([('file', 'f_op'),
                        ('file', 'private_data'),
                        ('mm_segment_t', 'seg')])},
        {'call': 'getdents64', 'reason': set([('fd', 'file')])},
        {'call': 'listen', 'reason': set([('fd', 'file')])},
        {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'get_robust_list',
         'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
        {'call': 'mq_timedsend',
         'reason': set([('fd', 'file'), ('mm_segment_t', 'seg')])},
        {'call': 'sched_getscheduler',
         'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
        {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'shmctl',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
        {'call': 'swapon',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'fallocate', 'reason': set([('fd', 'file')])},
        {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
        {'call': 'sched_getattr', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'sched_setscheduler',
         'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'llseek', 'reason': set([('fd', 'file')])},
        {'call': 'mmap_pgoff',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
        {'call': 'readv', 'reason': set([('fd', 'file')])},
        {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
        {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
        {'call': 'write', 'reason': set([('fd', 'file')])},
        {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
        {'call': 'sendfile', 'reason': set([('fd', 'file')])},
        {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'mq_open',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'sched_getparam', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'open_by_handle_at',
         'reason': set([('file', 'f_op'), ('file', 'private_data')])},
        {'call': 'bind', 'reason': set([('fd', 'file')])},
        {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
        {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
 'brk': [{'call': 'swapoff',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'remap_file_pages',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
```

```
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'io_getevents',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'migrate_pages',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'shmdt',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'brk',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'get_mempolicy',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'munlockall',
         'reason': set([('mm_struct', 'def_flags'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'pkey_mprotect',
         'reason': set([('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'madvise',
         'reason': set([('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'getrusage',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'io_setup',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'mprotect',
         'reason': set([('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'mremap',
```



```
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'io_destroy',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'mbind',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'prctl',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'move_pages',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'modify_ldt',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'munlock',
         'reason': set([('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'mincore',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'io_cancel',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'mlockall',
         'reason': set([('mm_struct', 'def_flags'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])}],
 'capset': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
            {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getaffinity',
```

```
             'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setaffinity',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
            {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
            {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
            {'call': 'perf_event_open',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
            {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
            {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
            {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
            {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
            {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
            {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getparam',
             'reason': set([('task_struct', 'cred')])}],
 'clock_adjtime': [{'call': 'clock_getres',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_delete',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_create',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'clock_gettime',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_settime',
```



```
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_gettime',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'clock_settime',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'clock_nanosleep',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'clock_adjtime',
                    'reason': set([('k_clock', 'clock_adj')])}],
 'clock_nanosleep': [{'call': 'rt_sigtimedwait',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'mq_unlink',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'swapoff',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'clock_getres',
                      'reason': set([('k_clock', 'nsleep')])},
                     {'call': 'timer_delete',
                      'reason': set([('k_clock', 'nsleep')])},
                     {'call': 'fchmod',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'memfd_create',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'readlinkat',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'io_getevents',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'fchown',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'mq_timedreceive',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'utime',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'semtimedop',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'settimeofday',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'timer_create',
                      'reason': set([('k_clock', 'nsleep')])},
                     {'call': 'clock_gettime',
                      'reason': set([('k_clock', 'nsleep')])},
                     {'call': 'sched_rr_get_interval',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'timerfd_gettime',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'pselect6',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'uselib',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'fchmodat',
```

```
654                      'reason': set([('timespec', 'tv_nsec'),
655                                     ('timespec', 'tv_sec')])},
656                    {'call': 'inotify_add_watch',
657                     'reason': set([('timespec', 'tv_nsec'),
658                                    ('timespec', 'tv_sec')])},
659                    {'call': 'timer_settime',
660                     'reason': set([('k_clock', 'nsleep'),
661                                    ('timespec', 'tv_nsec'),
662                                    ('timespec', 'tv_sec')])},
663                    {'call': 'ftruncate',
664                     'reason': set([('timespec', 'tv_nsec'),
665                                    ('timespec', 'tv_sec')])},
666                    {'call': 'timer_gettime',
667                     'reason': set([('k_clock', 'nsleep'),
668                                    ('timespec', 'tv_nsec'),
669                                    ('timespec', 'tv_sec')])},
670                    {'call': 'ioctl',
671                     'reason': set([('timespec', 'tv_nsec'),
672                                    ('timespec', 'tv_sec')])},
673                    {'call': 'linkat',
674                     'reason': set([('timespec', 'tv_nsec'),
675                                    ('timespec', 'tv_sec')])},
676                    {'call': 'stime',
677                     'reason': set([('timespec', 'tv_nsec'),
678                                    ('timespec', 'tv_sec')])},
679                    {'call': 'futimesat',
680                     'reason': set([('timespec', 'tv_nsec'),
681                                    ('timespec', 'tv_sec')])},
682                    {'call': 'poll',
683                     'reason': set([('timespec', 'tv_nsec'),
684                                    ('timespec', 'tv_sec')])},
685                    {'call': 'clock_settime',
686                     'reason': set([('k_clock', 'nsleep')])},
687                    {'call': 'select',
688                     'reason': set([('timespec', 'tv_nsec'),
689                                    ('timespec', 'tv_sec')])},
690                    {'call': 'unlink',
691                     'reason': set([('timespec', 'tv_nsec'),
692                                    ('timespec', 'tv_sec')])},
693                    {'call': 'nanosleep',
694                     'reason': set([('timespec', 'tv_nsec'),
695                                    ('timespec', 'tv_sec')])},
696                    {'call': 'mq_getsetattr',
697                     'reason': set([('timespec', 'tv_nsec'),
698                                    ('timespec', 'tv_sec')])},
699                    {'call': 'faccessat',
700                     'reason': set([('timespec', 'tv_nsec'),
701                                    ('timespec', 'tv_sec')])},
702                    {'call': 'mq_timedsend',
703                     'reason': set([('timespec', 'tv_nsec'),
704                                    ('timespec', 'tv_sec')])},
705                    {'call': 'swapon',
706                     'reason': set([('timespec', 'tv_nsec'),
707                                    ('timespec', 'tv_sec')])},
708                    {'call': 'epoll_wait',
709                     'reason': set([('timespec', 'tv_nsec'),
710                                    ('timespec', 'tv_sec')])},
711                    {'call': 'fchownat',
712                     'reason': set([('timespec', 'tv_nsec'),
713                                    ('timespec', 'tv_sec')])},
714                    {'call': 'fstat',
715                     'reason': set([('timespec', 'tv_nsec'),
716                                    ('timespec', 'tv_sec')])},
717                    {'call': 'timerfd_settime',
718                     'reason': set([('timespec', 'tv_nsec'),
719                                    ('timespec', 'tv_sec')])},
```



```
720                    {'call': 'mq_notify',
721                     'reason': set([('timespec', 'tv_nsec'),
722                                    ('timespec', 'tv_sec')])},
723                    {'call': 'sendfile',
724                     'reason': set([('timespec', 'tv_nsec'),
725                                    ('timespec', 'tv_sec')])},
726                    {'call': 'newfstat',
727                     'reason': set([('timespec', 'tv_nsec'),
728                                    ('timespec', 'tv_sec')])},
729                    {'call': 'clock_nanosleep',
730                     'reason': set([('k_clock', 'nsleep'),
731                                    ('timespec', 'tv_nsec'),
732                                    ('timespec', 'tv_sec')])},
733                    {'call': 'unlinkat',
734                     'reason': set([('timespec', 'tv_nsec'),
735                                    ('timespec', 'tv_sec')])},
736                    {'call': 'clock_adjtime',
737                     'reason': set([('k_clock', 'nsleep')])},
738                    {'call': 'futex',
739                     'reason': set([('timespec', 'tv_nsec'),
740                                    ('timespec', 'tv_sec')])},
741                    {'call': 'recvmmsg',
742                     'reason': set([('timespec', 'tv_nsec'),
743                                    ('timespec', 'tv_sec')])},
744                    {'call': 'sendfile64',
745                     'reason': set([('timespec', 'tv_nsec'),
746                                    ('timespec', 'tv_sec')])},
747                    {'call': 'ppoll',
748                     'reason': set([('timespec', 'tv_nsec'),
749                                    ('timespec', 'tv_sec')])}],
750  'clock_settime': [{'call': 'clock_getres',
751                     'reason': set([('k_clock', 'clock_set')])},
752                    {'call': 'timer_delete',
753                     'reason': set([('k_clock', 'clock_set')])},
754                    {'call': 'timer_create',
755                     'reason': set([('k_clock', 'clock_set')])},
756                    {'call': 'clock_gettime',
757                     'reason': set([('k_clock', 'clock_set')])},
758                    {'call': 'timer_settime',
759                     'reason': set([('k_clock', 'clock_set')])},
760                    {'call': 'timer_gettime',
761                     'reason': set([('k_clock', 'clock_set')])},
762                    {'call': 'clock_settime',
763                     'reason': set([('k_clock', 'clock_set')])},
764                    {'call': 'clock_nanosleep',
765                     'reason': set([('k_clock', 'clock_set')])},
766                    {'call': 'clock_adjtime',
767                     'reason': set([('k_clock', 'clock_set')])}],
768  'connect': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
769              {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
770              {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
771              {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
772              {'call': 'readahead', 'reason': set([('fd', 'file')])},
773              {'call': 'getdents', 'reason': set([('fd', 'file')])},
774              {'call': 'writev', 'reason': set([('fd', 'file')])},
775              {'call': 'preadv64', 'reason': set([('fd', 'file')])},
776              {'call': 'fchmod', 'reason': set([('fd', 'file')])},
777              {'call': 'pread64', 'reason': set([('fd', 'file')])},
778              {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
779              {'call': 'read', 'reason': set([('fd', 'file')])},
780              {'call': 'fchown', 'reason': set([('fd', 'file')])},
781              {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
782              {'call': 'utime', 'reason': set([('fd', 'file')])},
783              {'call': 'fsync', 'reason': set([('fd', 'file')])},
784              {'call': 'bpf', 'reason': set([('fd', 'file')])},
785              {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
```

```
786             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
787             {'call': 'sendto', 'reason': set([('fd', 'file')])},
788             {'call': 'tee', 'reason': set([('fd', 'file')])},
789             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
790             {'call': 'lseek', 'reason': set([('fd', 'file')])},
791             {'call': 'connect', 'reason': set([('fd', 'file')])},
792             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
793             {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
794             {'call': 'flock', 'reason': set([('fd', 'file')])},
795             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
796             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
797             {'call': 'accept4', 'reason': set([('fd', 'file')])},
798             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
799             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
800             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
801             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
802             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
803             {'call': 'splice', 'reason': set([('fd', 'file')])},
804             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
805             {'call': 'preadv', 'reason': set([('fd', 'file')])},
806             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
807             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
808             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
809             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
810             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
811             {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
812             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
813             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
814             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
815             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
816             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
817             {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
818             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
819             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
820             {'call': 'listen', 'reason': set([('fd', 'file')])},
821             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
822             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
823             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
824             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
825             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
826             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
827             {'call': 'llseek', 'reason': set([('fd', 'file')])},
828             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
829             {'call': 'readv', 'reason': set([('fd', 'file')])},
830             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
831             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
832             {'call': 'write', 'reason': set([('fd', 'file')])},
833             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
834             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
835             {'call': 'bind', 'reason': set([('fd', 'file')])},
836             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
837             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
838  'copy_file_range': [{'call': 'syncfs',
839                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
840                      {'call': 'vmsplice',
841                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
842                      {'call': 'pwritev64',
843                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
844                      {'call': 'fremovexattr',
845                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
846                      {'call': 'readahead',
847                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
848                      {'call': 'getdents',
849                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
850                      {'call': 'writev',
851                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
852                      {'call': 'preadv64',
853                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
854                      {'call': 'fchmod',
855                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
856                      {'call': 'pread64',
857                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
858                      {'call': 'signalfd4',
859                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
860                      {'call': 'read',
861                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
862                      {'call': 'fchown',
863                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
864                      {'call': 'mq_timedreceive',
865                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
866                      {'call': 'utime',
867                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
868                      {'call': 'fsync',
869                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
870                      {'call': 'bpf',
871                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
872                      {'call': 'recvfrom',
873                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
874                      {'call': 'fsetxattr',
875                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
876                      {'call': 'sendto',
877                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
878                      {'call': 'tee',
879                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
880                      {'call': 'sync_file_range',
881                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
882                      {'call': 'lseek',
883                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
884                      {'call': 'connect',
885                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
886                      {'call': 'getsockname',
887                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
888                      {'call': 'epoll_ctl',
889                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
890                      {'call': 'flock',
891                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
892                      {'call': 'pwritev',
893                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
894                      {'call': 'fchdir',
895                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
896                      {'call': 'accept4',
897                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
898                      {'call': 'old_readdir',
899                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
900                      {'call': 'inotify_rm_watch',
901                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
902                      {'call': 'utimensat',
903                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
904                      {'call': 'inotify_add_watch',
905                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
906                      {'call': 'preadv2',
907                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
908                      {'call': 'splice',
909                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
910                      {'call': 'ftruncate',
911                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
912                      {'call': 'preadv',
913                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
914                      {'call': 'getpeername',
915                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
916                      {'call': 'setsockopt',
917                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
                    {'call': 'fcntl',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'ioctl',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'pwrite64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'perf_event_open',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'pwritev64v2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'futimesat',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'pwritev2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'shutdown',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'getsockopt',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_getsetattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fdatasync',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'getdents64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'listen',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'copy_file_range',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fgetxattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fcntl64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fallocate',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'epoll_wait',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'llseek',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'preadv64v2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'readv',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fstatfs',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fstatfs64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'write',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_notify',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'sendfile',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'bind',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'flistxattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'sendfile64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])}],
'delete_module': [{'call': 'delete_module',
                   'reason': set([('module', 'exit'),
                                  ('module', 'init'),
                                  ('module', 'state')])},
                  {'call': 'init_module',
                   'reason': set([('module', 'exit'),
```



```
                                  ('module', 'init'),
                                  ('module', 'state')])},
                  {'call': 'finit_module',
                   'reason': set([('module', 'exit'),
                                  ('module', 'init'),
                                  ('module', 'state')])}],
'dmi_modalias_show': [{'call': 'dmi_modalias_show',
                       'reason': set([('mafield', 'field'),
                                      ('mafield', 'prefix')])}],
'dup3': [{'call': 'dup3',
          'reason': set([('fdtable', 'max_fds'),
                         ('files_struct', 'fdt'),
                         ('files_struct', 'resize_in_progress')])},
         {'call': 'unshare',
          'reason': set([('fdtable', 'max_fds'),
                         ('files_struct', 'fdt'),
                         ('files_struct', 'resize_in_progress')])},
         {'call': 'select', 'reason': set([('fdtable', 'max_fds')])},
         {'call': 'dup2',
          'reason': set([('fdtable', 'max_fds'),
                         ('files_struct', 'fdt'),
                         ('files_struct', 'resize_in_progress')])}],
'epoll_create1': [{'call': 'keyctl',
                   'reason': set([('cred', 'user'),
                                  ('task_struct', 'cred')])},
                  {'call': 'rt_sigtimedwait',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'setfsuid', 'reason': set([('cred', 'user')])},
                  {'call': 'msgrcv',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
                  {'call': 'getresuid16', 'reason': set([('cred', 'user')])},
                  {'call': 'getresgid', 'reason': set([('cred', 'user')])},
                  {'call': 'sched_getaffinity',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'sched_setparam',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'setgid', 'reason': set([('cred', 'user')])},
                  {'call': 'ioprio_set',
                   'reason': set([('cred', 'user'),
                                  ('task_struct', 'cred')])},
                  {'call': 'capset', 'reason': set([('cred', 'user')])},
                  {'call': 'getppid',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'mq_timedreceive',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'getresgid16', 'reason': set([('cred', 'user')])},
                  {'call': 'capget',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'sched_setaffinity',
                   'reason': set([('cred', 'user'),
                                  ('task_struct', 'cred')])},
                  {'call': 'setfsgid', 'reason': set([('cred', 'user')])},
                  {'call': 'unshare', 'reason': set([('cred', 'user')])},
                  {'call': 'signal',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'setreuid', 'reason': set([('cred', 'user')])},
                  {'call': 'semtimedop',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'umount',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'sched_rr_get_interval',
                   'reason': set([('task_struct', 'cred')])},
                  {'call': 'epoll_create1',
                   'reason': set([('cred', 'user'),
                                  ('eventpoll', 'user'),
```

```
                                    ('eventpoll', 'ws')])},
                    {'call': 'getresuid', 'reason': set([('cred', 'user')])},
                    {'call': 'epoll_ctl',
                     'reason': set([('eventpoll', 'user'),
                                    ('eventpoll', 'ws')])},
                    {'call': 'rt_sigprocmask',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'setsid',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'sigaltstack',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'sched_setattr',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'migrate_pages',
                     'reason': set([('cred', 'user'),
                                    ('task_struct', 'cred')])},
                    {'call': 'getitimer',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'setpgid',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'setresgid', 'reason': set([('cred', 'user')])},
                    {'call': 'setregid', 'reason': set([('cred', 'user')])},
                    {'call': 'getsid',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'prlimit64',
                     'reason': set([('cred', 'user'),
                                    ('task_struct', 'cred')])},
                    {'call': 'perf_event_open',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'getgroups16', 'reason': set([('cred', 'user')])},
                    {'call': 'rt_sigaction',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'getpgid',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'getpriority',
                     'reason': set([('cred', 'user'),
                                    ('task_struct', 'cred')])},
                    {'call': 'sigaction',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'faccessat', 'reason': set([('cred', 'user')])},
                    {'call': 'setns',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                    {'call': 'get_robust_list',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'ptrace',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'epoll_wait',
                     'reason': set([('eventpoll', 'user'),
                                    ('eventpoll', 'ws')])},
                    {'call': 'sched_getattr',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'getrusage',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'setresuid', 'reason': set([('cred', 'user')])},
                    {'call': 'setitimer',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'ioprio_get',
                     'reason': set([('cred', 'user'),
                                    ('task_struct', 'cred')])},
```



```
                    {'call': 'vfork',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'setuid', 'reason': set([('cred', 'user')])},
                    {'call': 'prctl',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'move_pages',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'getgroups', 'reason': set([('cred', 'user')])},
                    {'call': 'setpriority',
                     'reason': set([('cred', 'user'),
                                    ('task_struct', 'cred')])},
                    {'call': 'clone',
                     'reason': set([('task_struct', 'cred')])},
                    {'call': 'sched_getparam',
                     'reason': set([('task_struct', 'cred')])}],
 'epoll_ctl': [{'call': 'syncfs',
                'reason': set([('fd', 'file'),
                               ('fd', 'flags'),
                               ('list_head', 'next')])},
               {'call': 'keyctl', 'reason': set([('list_head', 'next')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('list_head', 'next')])},
               {'call': 'vmsplice',
                'reason': set([('fd', 'file'),
                               ('fd', 'flags'),
                               ('list_head', 'next')])},
               {'call': 'msgrcv', 'reason': set([('list_head', 'next')])},
               {'call': 'eventfd2',
                'reason': set([('file', 'f_op'), ('list_head', 'next')])},
               {'call': 'mq_unlink', 'reason': set([('list_head', 'next')])},
               {'call': 'pwritev64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'kill', 'reason': set([('list_head', 'next')])},
               {'call': 'swapoff',
                'reason': set([('file', 'f_op'), ('list_head', 'next')])},
               {'call': 'fremovexattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'readahead',
                'reason': set([('fd', 'file'),
                               ('fd', 'flags'),
                               ('list_head', 'next')])},
               {'call': 'getdents',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'timer_delete',
                'reason': set([('list_head', 'next')])},
               {'call': 'sched_getaffinity',
                'reason': set([('list_head', 'next')])},
               {'call': 'writev',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'sched_setparam',
                'reason': set([('list_head', 'next')])},
               {'call': 'fchmod',
                'reason': set([('fd', 'file'),
                               ('fd', 'flags'),
                               ('list_head', 'next')])},
               {'call': 'setgid', 'reason': set([('list_head', 'next')])},
               {'call': 'pread64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pivot_root', 'reason': set([('list_head', 'next')])},
               {'call': 'signalfd4',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'memfd_create',
                'reason': set([('file', 'f_op'), ('list_head', 'next')])},
               {'call': 'ioprio_set', 'reason': set([('list_head', 'next')])},
```

```
1182                {'call': 'delete_module',
1183                 'reason': set([('list_head', 'next')])},
1184                {'call': 'remap_file_pages',
1185                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1186                {'call': 'dup3',
1187                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1188                {'call': 'readlinkat', 'reason': set([('list_head', 'next')])},
1189                {'call': 'read',
1190                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1191                {'call': 'io_getevents',
1192                 'reason': set([('list_head', 'next')])},
1193                {'call': 'getppid', 'reason': set([('list_head', 'next')])},
1194                {'call': 'fchown',
1195                 'reason': set([('fd', 'file'),
1196                                ('fd', 'flags'),
1197                                ('list_head', 'next')])},
1198                {'call': 'mq_timedreceive',
1199                 'reason': set([('fd', 'file'),
1200                                ('fd', 'flags'),
1201                                ('list_head', 'next')])},
1202                {'call': 'capget', 'reason': set([('list_head', 'next')])},
1203                {'call': 'utime',
1204                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1205                {'call': 'sched_setaffinity',
1206                 'reason': set([('list_head', 'next')])},
1207                {'call': 'ustat', 'reason': set([('list_head', 'next')])},
1208                {'call': 'fsync',
1209                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1210                {'call': 'bpf',
1211                 'reason': set([('fd', 'file'),
1212                                ('fd', 'flags'),
1213                                ('list_head', 'next')])},
1214                {'call': 'unshare', 'reason': set([('list_head', 'next')])},
1215                {'call': 'signal', 'reason': set([('list_head', 'next')])},
1216                {'call': 'setreuid', 'reason': set([('list_head', 'next')])},
1217                {'call': 'semtimedop', 'reason': set([('list_head', 'next')])},
1218                {'call': 'umount', 'reason': set([('list_head', 'next')])},
1219                {'call': 'recvfrom',
1220                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1221                {'call': 'fsetxattr',
1222                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1223                {'call': 'timer_create',
1224                 'reason': set([('list_head', 'next')])},
1225                {'call': 'sendto',
1226                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1227                {'call': 'mkdirat', 'reason': set([('list_head', 'next')])},
1228                {'call': 'sched_rr_get_interval',
1229                 'reason': set([('list_head', 'next')])},
1230                {'call': 'epoll_create1',
1231                 'reason': set([('epitem', 'nwait'),
1232                                ('epitem', 'ws'),
1233                                ('epoll_event', 'events'),
1234                                ('file', 'f_op'),
1235                                ('list_head', 'next')])},
1236                {'call': 'timerfd_gettime',
1237                 'reason': set([('list_head', 'next')])},
1238                {'call': 'tee',
1239                 'reason': set([('fd', 'file'),
1240                                ('fd', 'flags'),
1241                                ('list_head', 'next')])},
1242                {'call': 'semctl', 'reason': set([('list_head', 'next')])},
1243                {'call': 'sync_file_range',
1244                 'reason': set([('fd', 'file'),
1245                                ('fd', 'flags'),
1246                                ('list_head', 'next')])},
1247                {'call': 'lseek',
```



```
1248                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1249                {'call': 'connect',
1250                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1251                {'call': 'getsockname',
1252                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1253                {'call': 'epoll_ctl',
1254                 'reason': set([('epitem', 'nwait'),
1255                                ('epitem', 'ws'),
1256                                ('epoll_event', 'events'),
1257                                ('fd', 'file'),
1258                                ('fd', 'flags'),
1259                                ('file', 'f_op'),
1260                                ('list_head', 'next')])},
1261                {'call': 'flock',
1262                 'reason': set([('fd', 'file'),
1263                                ('fd', 'flags'),
1264                                ('file', 'f_op'),
1265                                ('list_head', 'next')])},
1266                {'call': 'pwritev',
1267                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1268                {'call': 'fchdir',
1269                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1270                {'call': 'openat',
1271                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1272                {'call': 'lookup_dcookie',
1273                 'reason': set([('list_head', 'next')])},
1274                {'call': 'uselib',
1275                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1276                {'call': 'renameat2', 'reason': set([('list_head', 'next')])},
1277                {'call': 'rt_sigprocmask',
1278                 'reason': set([('list_head', 'next')])},
1279                {'call': 'accept4',
1280                 'reason': set([('fd', 'file'),
1281                                ('fd', 'flags'),
1282                                ('file', 'f_op'),
1283                                ('list_head', 'next')])},
1284                {'call': 'msgctl', 'reason': set([('list_head', 'next')])},
1285                {'call': 'reboot', 'reason': set([('list_head', 'next')])},
1286                {'call': 'setsid', 'reason': set([('list_head', 'next')])},
1287                {'call': 'set_trip_temp',
1288                 'reason': set([('list_head', 'next')])},
1289                {'call': 'sigaltstack',
1290                 'reason': set([('list_head', 'next')])},
1291                {'call': 'sched_setattr',
1292                 'reason': set([('list_head', 'next')])},
1293                {'call': 'old_readdir',
1294                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1295                {'call': 'inotify_rm_watch',
1296                 'reason': set([('fd', 'file'),
1297                                ('fd', 'flags'),
1298                                ('list_head', 'next')])},
1299                {'call': 'socketpair',
1300                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1301                {'call': 'utimensat',
1302                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1303                {'call': 'migrate_pages',
1304                 'reason': set([('list_head', 'next')])},
1305                {'call': 'getitimer', 'reason': set([('list_head', 'next')])},
1306                {'call': 'fchmodat', 'reason': set([('list_head', 'next')])},
1307                {'call': 'setpgid', 'reason': set([('list_head', 'next')])},
1308                {'call': 'init_module',
1309                 'reason': set([('list_head', 'next')])},
1310                {'call': 'setresgid', 'reason': set([('list_head', 'next')])},
1311                {'call': 'getcwd', 'reason': set([('list_head', 'next')])},
1312                {'call': 'inotify_add_watch',
1313                 'reason': set([('fd', 'file'),
```

```
                                ('fd', 'flags'),
                                ('list_head', 'next')])},
                {'call': 'get_trip_temp',
                 'reason': set([('list_head', 'next')])},
                {'call': 'preadv2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'timer_settime',
                 'reason': set([('list_head', 'next')])},
                {'call': 'setregid', 'reason': set([('list_head', 'next')])},
                {'call': 'splice',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('list_head', 'next')])},
                {'call': 'ftruncate',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('list_head', 'next')])},
                {'call': 'timer_gettime',
                 'reason': set([('list_head', 'next')])},
                {'call': 'preadv',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'getpeername',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'getsid', 'reason': set([('list_head', 'next')])},
                {'call': 'shmat',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'setsockopt',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mknodat', 'reason': set([('list_head', 'next')])},
                {'call': 'socket',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'symlinkat', 'reason': set([('list_head', 'next')])},
                {'call': 'pipe2',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'fcntl',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'ioctl',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('list_head', 'next')])},
                {'call': 'prlimit64', 'reason': set([('list_head', 'next')])},
                {'call': 'pwrite64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'perf_event_open',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('file', 'f_op'),
                                ('list_head', 'next')])},
                {'call': 'linkat', 'reason': set([('list_head', 'next')])},
                {'call': 'getgroups16',
                 'reason': set([('list_head', 'next')])},
                {'call': 'shmdt',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'pwritev64v2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'quotactl', 'reason': set([('list_head', 'next')])},
                {'call': 'rt_sigaction',
                 'reason': set([('list_head', 'next')])},
                {'call': 'futimesat',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'request_key',
                 'reason': set([('list_head', 'next')])},
                {'call': 'getpgid', 'reason': set([('list_head', 'next')])},
                {'call': 'brk', 'reason': set([('list_head', 'next')])},
                {'call': 'pwritev2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
                {'call': 'shutdown',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'acct',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'open',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'unlink', 'reason': set([('list_head', 'next')])},
                {'call': 'getsockopt',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'exit_group', 'reason': set([('list_head', 'next')])},
                {'call': 'getpriority',
                 'reason': set([('list_head', 'next')])},
                {'call': 'sigaction', 'reason': set([('list_head', 'next')])},
                {'call': 'mq_getsetattr',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('list_head', 'next')])},
                {'call': 'faccessat', 'reason': set([('list_head', 'next')])},
                {'call': 'rmdir', 'reason': set([('list_head', 'next')])},
                {'call': 'dup',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'fdatasync',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'setgroups16',
                 'reason': set([('list_head', 'next')])},
                {'call': 'setns',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'getdents64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'listen',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fork', 'reason': set([('list_head', 'next')])},
                {'call': 'get_mempolicy',
                 'reason': set([('list_head', 'next')])},
                {'call': 'io_submit', 'reason': set([('list_head', 'next')])},
                {'call': 'get_robust_list',
                 'reason': set([('list_head', 'next')])},
                {'call': 'copy_file_range',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mq_timedsend',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('list_head', 'next')])},
                {'call': 'sched_yield',
                 'reason': set([('list_head', 'next')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('list_head', 'next')])},
                {'call': 'fgetxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'ptrace', 'reason': set([('list_head', 'next')])},
                {'call': 'shmctl',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'fcntl64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'munlockall', 'reason': set([('list_head', 'next')])},
                {'call': 'swapon',
                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
                {'call': 'fallocate',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'pkey_mprotect',
                 'reason': set([('list_head', 'next')])},
                {'call': 'madvise', 'reason': set([('list_head', 'next')])},
                {'call': 'epoll_wait',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('list_head', 'next')])},
```

```
1446                {'call': 'sched_getattr',
1447                 'reason': set([('list_head', 'next')])},
1448                {'call': 'fchownat', 'reason': set([('list_head', 'next')])},
1449                {'call': 'getrusage', 'reason': set([('list_head', 'next')])},
1450                {'call': 'timerfd_settime',
1451                 'reason': set([('list_head', 'next')])},
1452                {'call': 'sched_setscheduler',
1453                 'reason': set([('list_head', 'next')])},
1454                {'call': 'setresuid', 'reason': set([('list_head', 'next')])},
1455                {'call': 'setitimer', 'reason': set([('list_head', 'next')])},
1456                {'call': 'ioprio_get', 'reason': set([('list_head', 'next')])},
1457                {'call': 'vfork', 'reason': set([('list_head', 'next')])},
1458                {'call': 'setuid', 'reason': set([('list_head', 'next')])},
1459                {'call': 'llseek',
1460                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1461                {'call': 'io_setup', 'reason': set([('list_head', 'next')])},
1462                {'call': 'mprotect', 'reason': set([('list_head', 'next')])},
1463                {'call': 'mmap_pgoff',
1464                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1465                {'call': 'mremap', 'reason': set([('list_head', 'next')])},
1466                {'call': 'io_destroy', 'reason': set([('list_head', 'next')])},
1467                {'call': 'mbind', 'reason': set([('list_head', 'next')])},
1468                {'call': 'preadv64v2',
1469                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1470                {'call': 'readv',
1471                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1472                {'call': 'prctl', 'reason': set([('list_head', 'next')])},
1473                {'call': 'move_pages', 'reason': set([('list_head', 'next')])},
1474                {'call': 'timerfd_create',
1475                 'reason': set([('list_head', 'next')])},
1476                {'call': 'fstatfs',
1477                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1478                {'call': 'modify_ldt', 'reason': set([('list_head', 'next')])},
1479                {'call': 'getgroups', 'reason': set([('list_head', 'next')])},
1480                {'call': 'fstatfs64',
1481                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1482                {'call': 'dup2', 'reason': set([('list_head', 'next')])},
1483                {'call': 'get_curr_temp',
1484                 'reason': set([('list_head', 'next')])},
1485                {'call': 'msgsnd', 'reason': set([('list_head', 'next')])},
1486                {'call': 'write',
1487                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1488                {'call': 'munlock', 'reason': set([('list_head', 'next')])},
1489                {'call': 'setpriority',
1490                 'reason': set([('list_head', 'next')])},
1491                {'call': 'inotify_init1',
1492                 'reason': set([('list_head', 'next')])},
1493                {'call': 'mincore', 'reason': set([('list_head', 'next')])},
1494                {'call': 'mq_notify',
1495                 'reason': set([('fd', 'file'),
1496                                ('fd', 'flags'),
1497                                ('list_head', 'next')])},
1498                {'call': 'sendfile',
1499                 'reason': set([('fd', 'file'),
1500                                ('fd', 'flags'),
1501                                ('list_head', 'next')])},
1502                {'call': 'timer_getoverrun',
1503                 'reason': set([('list_head', 'next')])},
1504                {'call': 'kexec_load', 'reason': set([('list_head', 'next')])},
1505                {'call': 'clone', 'reason': set([('list_head', 'next')])},
1506                {'call': 'mq_open',
1507                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1508                {'call': 'setgroups', 'reason': set([('list_head', 'next')])},
1509                {'call': 'unlinkat', 'reason': set([('list_head', 'next')])},
1510                {'call': 'sched_getparam',
1511                 'reason': set([('list_head', 'next')])},
```



```
1512                {'call': 'io_cancel', 'reason': set([('list_head', 'next')])},
1513                {'call': 'open_by_handle_at',
1514                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1515                {'call': 'bind',
1516                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1517                {'call': 'flistxattr',
1518                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1519                {'call': 'finit_module',
1520                 'reason': set([('list_head', 'next')])},
1521                {'call': 'sendfile64',
1522                 'reason': set([('fd', 'file'),
1523                                ('fd', 'flags'),
1524                                ('list_head', 'next')])},
1525                {'call': 'mlockall', 'reason': set([('list_head', 'next')])}],
1526  'epoll_wait': [{'call': 'syncfs',
1527                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1528                 {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
1529                 {'call': 'rt_sigtimedwait',
1530                  'reason': set([('mm_segment_t', 'seg')])},
1531                 {'call': 'vmsplice',
1532                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1533                 {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
1534                 {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
1535                 {'call': 'pwritev64',
1536                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1537                 {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
1538                 {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
1539                 {'call': 'fremovexattr',
1540                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1541                 {'call': 'readahead',
1542                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1543                 {'call': 'getdents',
1544                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1545                 {'call': 'sched_getaffinity',
1546                  'reason': set([('mm_segment_t', 'seg')])},
1547                 {'call': 'writev',
1548                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1549                 {'call': 'preadv64',
1550                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1551                 {'call': 'sched_setparam',
1552                  'reason': set([('mm_segment_t', 'seg')])},
1553                 {'call': 'fchmod',
1554                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1555                 {'call': 'pread64',
1556                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1557                 {'call': 'signalfd4',
1558                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1559                 {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
1560                 {'call': 'ioprio_set',
1561                  'reason': set([('mm_segment_t', 'seg')])},
1562                 {'call': 'remap_file_pages',
1563                  'reason': set([('file', 'f_op')])},
1564                 {'call': 'dup3', 'reason': set([('file', 'f_op')])},
1565                 {'call': 'read',
1566                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1567                 {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
1568                 {'call': 'fchown',
1569                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
1570                 {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
1571                 {'call': 'mq_timedreceive',
1572                  'reason': set([('fd', 'file'),
1573                                 ('fd', 'flags'),
1574                                 ('mm_segment_t', 'seg')])},
1575                 {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
1576                 {'call': 'utime',
1577                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
1578                {'call': 'sched_setaffinity',
1579                 'reason': set([('mm_segment_t', 'seg')])},
1580                {'call': 'fsync',
1581                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1582                {'call': 'bpf',
1583                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1584                {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
1585                {'call': 'semtimedop',
1586                 'reason': set([('mm_segment_t', 'seg')])},
1587                {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
1588                {'call': 'recvfrom',
1589                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1590                {'call': 'fsetxattr',
1591                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1592                {'call': 'sendto',
1593                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1594                {'call': 'sched_rr_get_interval',
1595                 'reason': set([('mm_segment_t', 'seg')])},
1596                {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
1597                {'call': 'tee',
1598                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1599                {'call': 'sync_file_range',
1600                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1601                {'call': 'lseek',
1602                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1603                {'call': 'connect',
1604                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1605                {'call': 'getsockname',
1606                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1607                {'call': 'epoll_ctl',
1608                 'reason': set([('fd', 'file'),
1609                                ('fd', 'flags'),
1610                                ('file', 'f_op')])},
1611                {'call': 'flock',
1612                 'reason': set([('fd', 'file'),
1613                                ('fd', 'flags'),
1614                                ('file', 'f_op')])},
1615                {'call': 'pwritev',
1616                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1617                {'call': 'fchdir',
1618                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1619                {'call': 'openat', 'reason': set([('file', 'f_op')])},
1620                {'call': 'uselib', 'reason': set([('file', 'f_op')])},
1621                {'call': 'rt_sigprocmask',
1622                 'reason': set([('mm_segment_t', 'seg')])},
1623                {'call': 'accept4',
1624                 'reason': set([('fd', 'file'),
1625                                ('fd', 'flags'),
1626                                ('file', 'f_op')])},
1627                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
1628                {'call': 'sigaltstack',
1629                 'reason': set([('mm_segment_t', 'seg')])},
1630                {'call': 'sched_setattr',
1631                 'reason': set([('mm_segment_t', 'seg')])},
1632                {'call': 'old_readdir',
1633                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1634                {'call': 'inotify_rm_watch',
1635                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1636                {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
1637                {'call': 'utimensat',
1638                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1639                {'call': 'migrate_pages',
1640                 'reason': set([('mm_segment_t', 'seg')])},
1641                {'call': 'getitimer',
1642                 'reason': set([('mm_segment_t', 'seg')])},
1643                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
```



```
1644                {'call': 'inotify_add_watch',
1645                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1646                {'call': 'preadv2',
1647                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1648                {'call': 'splice',
1649                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1650                {'call': 'ftruncate',
1651                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1652                {'call': 'preadv',
1653                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1654                {'call': 'getpeername',
1655                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1656                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
1657                {'call': 'shmat', 'reason': set([('file', 'f_op')])},
1658                {'call': 'setsockopt',
1659                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1660                {'call': 'socket', 'reason': set([('file', 'f_op')])},
1661                {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
1662                {'call': 'fcntl',
1663                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1664                {'call': 'ioctl',
1665                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1666                {'call': 'prlimit64',
1667                 'reason': set([('mm_segment_t', 'seg')])},
1668                {'call': 'pwrite64',
1669                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1670                {'call': 'perf_event_open',
1671                 'reason': set([('fd', 'file'),
1672                                ('fd', 'flags'),
1673                                ('file', 'f_op'),
1674                                ('mm_segment_t', 'seg')])},
1675                {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
1676                {'call': 'pwritev64v2',
1677                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1678                {'call': 'rt_sigaction',
1679                 'reason': set([('mm_segment_t', 'seg')])},
1680                {'call': 'futimesat',
1681                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1682                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
1683                {'call': 'pwritev2',
1684                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1685                {'call': 'shutdown',
1686                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1687                {'call': 'acct', 'reason': set([('file', 'f_op')])},
1688                {'call': 'open', 'reason': set([('file', 'f_op')])},
1689                {'call': 'getsockopt',
1690                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1691                {'call': 'getpriority',
1692                 'reason': set([('mm_segment_t', 'seg')])},
1693                {'call': 'sigaction',
1694                 'reason': set([('mm_segment_t', 'seg')])},
1695                {'call': 'mq_getsetattr',
1696                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1697                {'call': 'dup', 'reason': set([('file', 'f_op')])},
1698                {'call': 'fdatasync',
1699                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1700                {'call': 'setns',
1701                 'reason': set([('file', 'f_op'), ('mm_segment_t', 'seg')])},
1702                {'call': 'getdents64',
1703                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1704                {'call': 'listen',
1705                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1706                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
1707                {'call': 'get_robust_list',
1708                 'reason': set([('mm_segment_t', 'seg')])},
1709                {'call': 'copy_file_range',
```

```
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mq_timedsend',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('mm_segment_t', 'seg')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fgetxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
                {'call': 'fcntl64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'swapon', 'reason': set([('file', 'f_op')])},
                {'call': 'fallocate',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'epoll_wait',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sched_getattr',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'getrusage',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'setitimer',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'ioprio_get',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'llseek',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
                {'call': 'preadv64v2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'readv',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'move_pages',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fstatfs',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fstatfs64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'write',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'setpriority',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_notify',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sendfile',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
                {'call': 'sched_getparam',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_op')])},
                {'call': 'bind',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'flistxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sendfile64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
 'faccessat': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('task_struct', 'cred')])},
```



```
               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
               {'call': 'eventfd2', 'reason': set([('path', 'mnt')])},
               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
               {'call': 'swapoff', 'reason': set([('path', 'mnt')])},
               {'call': 'sched_getaffinity',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setparam',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'pivot_root', 'reason': set([('path', 'mnt')])},
               {'call': 'memfd_create', 'reason': set([('path', 'mnt')])},
               {'call': 'ioprio_set',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'remap_file_pages', 'reason': set([('path', 'mnt')])},
               {'call': 'dup3', 'reason': set([('path', 'mnt')])},
               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedreceive',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setaffinity',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'unshare', 'reason': set([('path', 'mnt')])},
               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
               {'call': 'setreuid', 'reason': set([('cred', 'uid')])},
               {'call': 'semtimedop',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_rr_get_interval',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'epoll_create1', 'reason': set([('path', 'mnt')])},
               {'call': 'epoll_ctl', 'reason': set([('path', 'mnt')])},
               {'call': 'flock', 'reason': set([('path', 'mnt')])},
               {'call': 'openat', 'reason': set([('path', 'mnt')])},
               {'call': 'lookup_dcookie', 'reason': set([('path', 'mnt')])},
               {'call': 'uselib', 'reason': set([('path', 'mnt')])},
               {'call': 'rt_sigprocmask',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'accept4', 'reason': set([('path', 'mnt')])},
               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaltstack',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'socketpair', 'reason': set([('path', 'mnt')])},
               {'call': 'migrate_pages',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'getcwd', 'reason': set([('path', 'mnt')])},
               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'shmat', 'reason': set([('path', 'mnt')])},
               {'call': 'socket', 'reason': set([('path', 'mnt')])},
               {'call': 'pipe2', 'reason': set([('path', 'mnt')])},
               {'call': 'prlimit64',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'perf_event_open',
                'reason': set([('path', 'mnt'), ('task_struct', 'cred')])},
               {'call': 'shmdt', 'reason': set([('path', 'mnt')])},
               {'call': 'quotactl', 'reason': set([('path', 'mnt')])},
               {'call': 'rt_sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'acct', 'reason': set([('path', 'mnt')])},
               {'call': 'open', 'reason': set([('path', 'mnt')])},
               {'call': 'getpriority',
                'reason': set([('task_struct', 'cred')])},
```

```
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'dup', 'reason': set([('path', 'mnt')])},
                {'call': 'setns',
                 'reason': set([('path', 'mnt'), ('task_struct', 'cred')])},
                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
                {'call': 'shmctl', 'reason': set([('path', 'mnt')])},
                {'call': 'swapon', 'reason': set([('path', 'mnt')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setresuid', 'reason': set([('cred', 'uid')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setuid', 'reason': set([('cred', 'uid')])},
                {'call': 'mmap_pgoff', 'reason': set([('path', 'mnt')])},
                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_open', 'reason': set([('path', 'mnt')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('path', 'mnt')])}],
 'fallocate': [{'call': 'syncfs',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'vmsplice',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fremovexattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'readahead',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getdents',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'writev',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fchmod',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pread64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'signalfd4',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'read',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fchown',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
               {'call': 'mq_timedreceive',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'utime',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fsync',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'bpf',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'recvfrom',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fsetxattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'sendto',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'tee',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'sync_file_range',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'lseek',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'connect',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getsockname',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'epoll_ctl',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'flock',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fchdir',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'accept4',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'old_readdir',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'inotify_rm_watch',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'utimensat',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'inotify_add_watch',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'splice',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'ftruncate',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getpeername',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'setsockopt',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fcntl',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'ioctl',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwrite64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'perf_event_open',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev64v2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'futimesat',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
1974                {'call': 'pwritev2',
1975                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1976                {'call': 'shutdown',
1977                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1978                {'call': 'getsockopt',
1979                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1980                {'call': 'mq_getsetattr',
1981                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1982                {'call': 'fdatasync',
1983                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1984                {'call': 'getdents64',
1985                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1986                {'call': 'listen',
1987                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1988                {'call': 'copy_file_range',
1989                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1990                {'call': 'mq_timedsend',
1991                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1992                {'call': 'fgetxattr',
1993                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1994                {'call': 'fcntl64',
1995                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1996                {'call': 'fallocate',
1997                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
1998                {'call': 'epoll_wait',
1999                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2000                {'call': 'llseek',
2001                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2002                {'call': 'preadv64v2',
2003                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2004                {'call': 'readv',
2005                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2006                {'call': 'fstatfs',
2007                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2008                {'call': 'fstatfs64',
2009                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2010                {'call': 'write',
2011                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2012                {'call': 'mq_notify',
2013                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2014                {'call': 'sendfile',
2015                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2016                {'call': 'bind',
2017                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2018                {'call': 'flistxattr',
2019                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2020                {'call': 'sendfile64',
2021                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
2022  'fchdir': [{'call': 'syncfs',
2023              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2024             {'call': 'vmsplice',
2025              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2026             {'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
2027             {'call': 'pwritev64',
2028              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2029             {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
2030             {'call': 'fremovexattr',
2031              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2032             {'call': 'readahead',
2033              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2034             {'call': 'getdents',
2035              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2036             {'call': 'writev',
2037              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2038             {'call': 'preadv64',
2039              'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
2040             {'call': 'fchmod',
2041              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2042             {'call': 'pread64',
2043              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2044             {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
2045             {'call': 'signalfd4',
2046              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2047             {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
2048             {'call': 'remap_file_pages', 'reason': set([('path', 'dentry')])},
2049             {'call': 'dup3', 'reason': set([('path', 'dentry')])},
2050             {'call': 'read',
2051              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2052             {'call': 'fchown',
2053              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2054             {'call': 'mq_timedreceive',
2055              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2056             {'call': 'utime',
2057              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2058             {'call': 'fsync',
2059              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2060             {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2061             {'call': 'unshare', 'reason': set([('path', 'dentry')])},
2062             {'call': 'recvfrom',
2063              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2064             {'call': 'fsetxattr',
2065              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2066             {'call': 'sendto',
2067              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2068             {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
2069             {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2070             {'call': 'sync_file_range',
2071              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2072             {'call': 'lseek',
2073              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2074             {'call': 'connect',
2075              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2076             {'call': 'getsockname',
2077              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2078             {'call': 'epoll_ctl',
2079              'reason': set([('fd', 'file'),
2080                             ('fd', 'flags'),
2081                             ('path', 'dentry')])},
2082             {'call': 'flock',
2083              'reason': set([('fd', 'file'),
2084                             ('fd', 'flags'),
2085                             ('path', 'dentry')])},
2086             {'call': 'pwritev',
2087              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2088             {'call': 'fchdir',
2089              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2090             {'call': 'openat', 'reason': set([('path', 'dentry')])},
2091             {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
2092             {'call': 'uselib', 'reason': set([('path', 'dentry')])},
2093             {'call': 'accept4',
2094              'reason': set([('fd', 'file'),
2095                             ('fd', 'flags'),
2096                             ('path', 'dentry')])},
2097             {'call': 'old_readdir',
2098              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2099             {'call': 'inotify_rm_watch',
2100              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2101             {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
2102             {'call': 'utimensat',
2103              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2104             {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
2105             {'call': 'inotify_add_watch',
```

```
2106              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2107             {'call': 'preadv2',
2108              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2109             {'call': 'splice',
2110              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2111             {'call': 'ftruncate',
2112              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2113             {'call': 'preadv',
2114              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2115             {'call': 'getpeername',
2116              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2117             {'call': 'shmat', 'reason': set([('path', 'dentry')])},
2118             {'call': 'setsockopt',
2119              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2120             {'call': 'socket', 'reason': set([('path', 'dentry')])},
2121             {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
2122             {'call': 'fcntl',
2123              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2124             {'call': 'ioctl',
2125              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2126             {'call': 'pwrite64',
2127              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2128             {'call': 'perf_event_open',
2129              'reason': set([('fd', 'file'),
2130                             ('fd', 'flags'),
2131                             ('path', 'dentry')])},
2132             {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
2133             {'call': 'pwritev64v2',
2134              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2135             {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
2136             {'call': 'futimesat',
2137              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2138             {'call': 'pwritev2',
2139              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2140             {'call': 'shutdown',
2141              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2142             {'call': 'acct', 'reason': set([('path', 'dentry')])},
2143             {'call': 'open', 'reason': set([('path', 'dentry')])},
2144             {'call': 'getsockopt',
2145              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2146             {'call': 'mq_getsetattr',
2147              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2148             {'call': 'dup', 'reason': set([('path', 'dentry')])},
2149             {'call': 'fdatasync',
2150              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2151             {'call': 'setns', 'reason': set([('path', 'dentry')])},
2152             {'call': 'getdents64',
2153              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2154             {'call': 'listen',
2155              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2156             {'call': 'copy_file_range',
2157              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2158             {'call': 'mq_timedsend',
2159              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2160             {'call': 'fgetxattr',
2161              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2162             {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
2163             {'call': 'fcntl64',
2164              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2165             {'call': 'swapon', 'reason': set([('path', 'dentry')])},
2166             {'call': 'fallocate',
2167              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2168             {'call': 'epoll_wait',
2169              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2170             {'call': 'llseek',
2171              'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
2172             {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
2173             {'call': 'preadv64v2',
2174              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2175             {'call': 'readv',
2176              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2177             {'call': 'fstatfs',
2178              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2179             {'call': 'fstatfs64',
2180              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2181             {'call': 'write',
2182              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2183             {'call': 'mq_notify',
2184              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2185             {'call': 'sendfile',
2186              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2187             {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
2188             {'call': 'open_by_handle_at',
2189              'reason': set([('path', 'dentry')])},
2190             {'call': 'bind',
2191              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2192             {'call': 'flistxattr',
2193              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2194             {'call': 'sendfile64',
2195              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
2196  'fchmod': [{'call': 'syncfs',
2197              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2198             {'call': 'vmsplice',
2199              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2200             {'call': 'eventfd2',
2201              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2202             {'call': 'pwritev64',
2203              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2204             {'call': 'swapoff',
2205              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2206             {'call': 'fremovexattr',
2207              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2208             {'call': 'readahead',
2209              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2210             {'call': 'getdents',
2211              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2212             {'call': 'writev',
2213              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2214             {'call': 'preadv64',
2215              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2216             {'call': 'fchmod',
2217              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2218             {'call': 'pread64',
2219              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2220             {'call': 'pivot_root',
2221              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2222             {'call': 'signalfd4',
2223              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2224             {'call': 'memfd_create',
2225              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2226             {'call': 'remap_file_pages',
2227              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2228             {'call': 'dup3',
2229              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2230             {'call': 'read',
2231              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2232             {'call': 'fchown',
2233              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2234             {'call': 'mq_timedreceive',
2235              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2236             {'call': 'utime',
2237              'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
2238            {'call': 'fsync',
2239             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2240            {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2241            {'call': 'unshare',
2242             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2243            {'call': 'recvfrom',
2244             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2245            {'call': 'fsetxattr',
2246             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2247            {'call': 'sendto',
2248             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2249            {'call': 'epoll_create1',
2250             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2251            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2252            {'call': 'sync_file_range',
2253             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2254            {'call': 'lseek',
2255             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2256            {'call': 'connect',
2257             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2258            {'call': 'getsockname',
2259             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2260            {'call': 'epoll_ctl',
2261             'reason': set([('fd', 'file'),
2262                            ('fd', 'flags'),
2263                            ('path', 'dentry'),
2264                            ('path', 'mnt')])},
2265            {'call': 'flock',
2266             'reason': set([('fd', 'file'),
2267                            ('fd', 'flags'),
2268                            ('path', 'dentry'),
2269                            ('path', 'mnt')])},
2270            {'call': 'pwritev',
2271             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2272            {'call': 'fchdir',
2273             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2274            {'call': 'openat',
2275             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2276            {'call': 'lookup_dcookie',
2277             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2278            {'call': 'uselib',
2279             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2280            {'call': 'accept4',
2281             'reason': set([('fd', 'file'),
2282                            ('fd', 'flags'),
2283                            ('path', 'dentry'),
2284                            ('path', 'mnt')])},
2285            {'call': 'old_readdir',
2286             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2287            {'call': 'inotify_rm_watch',
2288             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2289            {'call': 'socketpair',
2290             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2291            {'call': 'utimensat',
2292             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2293            {'call': 'getcwd',
2294             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2295            {'call': 'inotify_add_watch',
2296             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2297            {'call': 'preadv2',
2298             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2299            {'call': 'splice',
2300             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2301            {'call': 'ftruncate',
2302             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2303            {'call': 'preadv',
```



```
2304             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2305            {'call': 'getpeername',
2306             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2307            {'call': 'shmat',
2308             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2309            {'call': 'setsockopt',
2310             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2311            {'call': 'socket',
2312             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2313            {'call': 'pipe2',
2314             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2315            {'call': 'fcntl',
2316             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2317            {'call': 'ioctl',
2318             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2319            {'call': 'pwrite64',
2320             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2321            {'call': 'perf_event_open',
2322             'reason': set([('fd', 'file'),
2323                            ('fd', 'flags'),
2324                            ('path', 'dentry'),
2325                            ('path', 'mnt')])},
2326            {'call': 'shmdt',
2327             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2328            {'call': 'pwritev64v2',
2329             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2330            {'call': 'quotactl',
2331             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2332            {'call': 'futimesat',
2333             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2334            {'call': 'pwritev2',
2335             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2336            {'call': 'shutdown',
2337             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2338            {'call': 'acct',
2339             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2340            {'call': 'open',
2341             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2342            {'call': 'getsockopt',
2343             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2344            {'call': 'mq_getsetattr',
2345             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2346            {'call': 'dup',
2347             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2348            {'call': 'fdatasync',
2349             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2350            {'call': 'setns',
2351             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2352            {'call': 'getdents64',
2353             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2354            {'call': 'listen',
2355             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2356            {'call': 'copy_file_range',
2357             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2358            {'call': 'mq_timedsend',
2359             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2360            {'call': 'fgetxattr',
2361             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2362            {'call': 'shmctl',
2363             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2364            {'call': 'fcntl64',
2365             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2366            {'call': 'swapon',
2367             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2368            {'call': 'fallocate',
2369             'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
2370             {'call': 'epoll_wait',
2371              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2372             {'call': 'llseek',
2373              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2374             {'call': 'mmap_pgoff',
2375              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2376             {'call': 'preadv64v2',
2377              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2378             {'call': 'readv',
2379              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2380             {'call': 'fstatfs',
2381              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2382             {'call': 'fstatfs64',
2383              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2384             {'call': 'write',
2385              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2386             {'call': 'mq_notify',
2387              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2388             {'call': 'sendfile',
2389              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2390             {'call': 'mq_open',
2391              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2392             {'call': 'open_by_handle_at',
2393              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2394             {'call': 'bind',
2395              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2396             {'call': 'flistxattr',
2397              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2398             {'call': 'sendfile64',
2399              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
2400  'fchmodat': [{'call': 'eventfd2',
2401                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2402               {'call': 'swapoff',
2403                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2404               {'call': 'pivot_root',
2405                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2406               {'call': 'memfd_create',
2407                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2408               {'call': 'remap_file_pages',
2409                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2410               {'call': 'dup3',
2411                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2412               {'call': 'unshare',
2413                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2414               {'call': 'epoll_create1',
2415                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2416               {'call': 'epoll_ctl',
2417                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2418               {'call': 'flock',
2419                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2420               {'call': 'openat',
2421                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2422               {'call': 'lookup_dcookie',
2423                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2424               {'call': 'uselib',
2425                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2426               {'call': 'accept4',
2427                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2428               {'call': 'socketpair',
2429                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2430               {'call': 'getcwd',
2431                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2432               {'call': 'shmat',
2433                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2434               {'call': 'socket',
2435                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```



```
2436               {'call': 'pipe2',
2437                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2438               {'call': 'perf_event_open',
2439                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2440               {'call': 'shmdt',
2441                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2442               {'call': 'quotactl',
2443                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2444               {'call': 'acct',
2445                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2446               {'call': 'open',
2447                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2448               {'call': 'dup',
2449                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2450               {'call': 'setns',
2451                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2452               {'call': 'shmctl',
2453                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2454               {'call': 'swapon',
2455                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2456               {'call': 'mmap_pgoff',
2457                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2458               {'call': 'mq_open',
2459                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
2460               {'call': 'open_by_handle_at',
2461                'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
2462  'fchown': [{'call': 'syncfs',
2463              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2464             {'call': 'vmsplice',
2465              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2466             {'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
2467             {'call': 'pwritev64',
2468              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2469             {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
2470             {'call': 'fremovexattr',
2471              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2472             {'call': 'readahead',
2473              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2474             {'call': 'getdents',
2475              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2476             {'call': 'writev',
2477              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2478             {'call': 'preadv64',
2479              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2480             {'call': 'fchmod',
2481              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2482             {'call': 'pread64',
2483              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2484             {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
2485             {'call': 'signalfd4',
2486              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2487             {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
2488             {'call': 'remap_file_pages', 'reason': set([('path', 'dentry')])},
2489             {'call': 'dup3', 'reason': set([('path', 'dentry')])},
2490             {'call': 'read',
2491              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2492             {'call': 'fchown',
2493              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2494             {'call': 'mq_timedreceive',
2495              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2496             {'call': 'utime',
2497              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2498             {'call': 'fsync',
2499              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2500             {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2501             {'call': 'unshare', 'reason': set([('path', 'dentry')])},
```

```
2502            {'call': 'recvfrom',
2503             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2504            {'call': 'fsetxattr',
2505             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2506            {'call': 'sendto',
2507             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2508            {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
2509            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2510            {'call': 'sync_file_range',
2511             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2512            {'call': 'lseek',
2513             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2514            {'call': 'connect',
2515             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2516            {'call': 'getsockname',
2517             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2518            {'call': 'epoll_ctl',
2519             'reason': set([('fd', 'file'),
2520                            ('fd', 'flags'),
2521                            ('path', 'dentry')])},
2522            {'call': 'flock',
2523             'reason': set([('fd', 'file'),
2524                            ('fd', 'flags'),
2525                            ('path', 'dentry')])},
2526            {'call': 'pwritev',
2527             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2528            {'call': 'fchdir',
2529             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2530            {'call': 'openat', 'reason': set([('path', 'dentry')])},
2531            {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
2532            {'call': 'uselib', 'reason': set([('path', 'dentry')])},
2533            {'call': 'accept4',
2534             'reason': set([('fd', 'file'),
2535                            ('fd', 'flags'),
2536                            ('path', 'dentry')])},
2537            {'call': 'old_readdir',
2538             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2539            {'call': 'inotify_rm_watch',
2540             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2541            {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
2542            {'call': 'utimensat',
2543             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2544            {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
2545            {'call': 'inotify_add_watch',
2546             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2547            {'call': 'preadv2',
2548             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2549            {'call': 'splice',
2550             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2551            {'call': 'ftruncate',
2552             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2553            {'call': 'preadv',
2554             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2555            {'call': 'getpeername',
2556             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2557            {'call': 'shmat', 'reason': set([('path', 'dentry')])},
2558            {'call': 'setsockopt',
2559             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2560            {'call': 'socket', 'reason': set([('path', 'dentry')])},
2561            {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
2562            {'call': 'fcntl',
2563             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2564            {'call': 'ioctl',
2565             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2566            {'call': 'pwrite64',
2567             'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
2568            {'call': 'perf_event_open',
2569             'reason': set([('fd', 'file'),
2570                            ('fd', 'flags'),
2571                            ('path', 'dentry')])},
2572            {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
2573            {'call': 'pwritev64v2',
2574             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2575            {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
2576            {'call': 'futimesat',
2577             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2578            {'call': 'pwritev2',
2579             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2580            {'call': 'shutdown',
2581             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2582            {'call': 'acct', 'reason': set([('path', 'dentry')])},
2583            {'call': 'open', 'reason': set([('path', 'dentry')])},
2584            {'call': 'getsockopt',
2585             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2586            {'call': 'mq_getsetattr',
2587             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2588            {'call': 'dup', 'reason': set([('path', 'dentry')])},
2589            {'call': 'fdatasync',
2590             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2591            {'call': 'setns', 'reason': set([('path', 'dentry')])},
2592            {'call': 'getdents64',
2593             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2594            {'call': 'listen',
2595             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2596            {'call': 'copy_file_range',
2597             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2598            {'call': 'mq_timedsend',
2599             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2600            {'call': 'fgetxattr',
2601             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2602            {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
2603            {'call': 'fcntl64',
2604             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2605            {'call': 'swapon', 'reason': set([('path', 'dentry')])},
2606            {'call': 'fallocate',
2607             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2608            {'call': 'epoll_wait',
2609             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2610            {'call': 'llseek',
2611             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2612            {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
2613            {'call': 'preadv64v2',
2614             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2615            {'call': 'readv',
2616             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2617            {'call': 'fstatfs',
2618             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2619            {'call': 'fstatfs64',
2620             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2621            {'call': 'write',
2622             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2623            {'call': 'mq_notify',
2624             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2625            {'call': 'sendfile',
2626             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2627            {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
2628            {'call': 'open_by_handle_at',
2629             'reason': set([('path', 'dentry')])},
2630            {'call': 'bind',
2631             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2632            {'call': 'flistxattr',
2633             'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
2634             {'call': 'sendfile64',
2635              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
2636  'fchownat': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
2637               {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
2638               {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
2639               {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
2640               {'call': 'remap_file_pages',
2641                'reason': set([('path', 'dentry')])},
2642               {'call': 'dup3', 'reason': set([('path', 'dentry')])},
2643               {'call': 'unshare', 'reason': set([('path', 'dentry')])},
2644               {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
2645               {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
2646               {'call': 'flock', 'reason': set([('path', 'dentry')])},
2647               {'call': 'openat', 'reason': set([('path', 'dentry')])},
2648               {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
2649               {'call': 'uselib', 'reason': set([('path', 'dentry')])},
2650               {'call': 'accept4', 'reason': set([('path', 'dentry')])},
2651               {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
2652               {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
2653               {'call': 'shmat', 'reason': set([('path', 'dentry')])},
2654               {'call': 'socket', 'reason': set([('path', 'dentry')])},
2655               {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
2656               {'call': 'perf_event_open',
2657                'reason': set([('path', 'dentry')])},
2658               {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
2659               {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
2660               {'call': 'acct', 'reason': set([('path', 'dentry')])},
2661               {'call': 'open', 'reason': set([('path', 'dentry')])},
2662               {'call': 'dup', 'reason': set([('path', 'dentry')])},
2663               {'call': 'setns', 'reason': set([('path', 'dentry')])},
2664               {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
2665               {'call': 'swapon', 'reason': set([('path', 'dentry')])},
2666               {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
2667               {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
2668               {'call': 'open_by_handle_at',
2669                'reason': set([('path', 'dentry')])}],
2670  'fcntl': [{'call': 'syncfs',
2671             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2672            {'call': 'vmsplice',
2673             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2674            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
2675            {'call': 'pwritev64',
2676             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2677            {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
2678            {'call': 'fremovexattr',
2679             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2680            {'call': 'readahead',
2681             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2682            {'call': 'getdents',
2683             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2684            {'call': 'writev',
2685             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2686            {'call': 'preadv64',
2687             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2688            {'call': 'fchmod',
2689             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2690            {'call': 'pread64',
2691             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2692            {'call': 'signalfd4',
2693             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2694            {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
2695            {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
2696            {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
2697            {'call': 'read', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2698            {'call': 'fchown',
2699             'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
2700            {'call': 'mq_timedreceive',
2701             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2702            {'call': 'utime',
2703             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2704            {'call': 'fsync',
2705             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2706            {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2707            {'call': 'recvfrom',
2708             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2709            {'call': 'fsetxattr',
2710             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2711            {'call': 'sendto',
2712             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2713            {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
2714            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
2715            {'call': 'sync_file_range',
2716             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2717            {'call': 'lseek',
2718             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2719            {'call': 'connect',
2720             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2721            {'call': 'getsockname',
2722             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2723            {'call': 'epoll_ctl',
2724             'reason': set([('fd', 'file'),
2725                            ('fd', 'flags'),
2726                            ('file', 'f_mode')])},
2727            {'call': 'flock',
2728             'reason': set([('fd', 'file'),
2729                            ('fd', 'flags'),
2730                            ('file', 'f_mode')])},
2731            {'call': 'pwritev',
2732             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2733            {'call': 'fchdir',
2734             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2735            {'call': 'openat', 'reason': set([('file', 'f_mode')])},
2736            {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
2737            {'call': 'accept4',
2738             'reason': set([('fd', 'file'),
2739                            ('fd', 'flags'),
2740                            ('file', 'f_mode')])},
2741            {'call': 'old_readdir',
2742             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2743            {'call': 'inotify_rm_watch',
2744             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2745            {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
2746            {'call': 'utimensat',
2747             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2748            {'call': 'inotify_add_watch',
2749             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2750            {'call': 'preadv2',
2751             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2752            {'call': 'splice',
2753             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2754            {'call': 'ftruncate',
2755             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2756            {'call': 'preadv',
2757             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2758            {'call': 'getpeername',
2759             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2760            {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
2761            {'call': 'setsockopt',
2762             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2763            {'call': 'socket', 'reason': set([('file', 'f_mode')])},
2764            {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
2765            {'call': 'fcntl',
```

```
2766             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2767            {'call': 'ioctl',
2768             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2769            {'call': 'pwrite64',
2770             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2771            {'call': 'perf_event_open',
2772             'reason': set([('fd', 'file'),
2773                            ('fd', 'flags'),
2774                            ('file', 'f_mode')])},
2775            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
2776            {'call': 'pwritev64v2',
2777             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2778            {'call': 'futimesat',
2779             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2780            {'call': 'pwritev2',
2781             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2782            {'call': 'shutdown',
2783             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2784            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
2785            {'call': 'open', 'reason': set([('file', 'f_mode')])},
2786            {'call': 'getsockopt',
2787             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2788            {'call': 'mq_getsetattr',
2789             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2790            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
2791            {'call': 'fdatasync',
2792             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2793            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
2794            {'call': 'getdents64',
2795             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2796            {'call': 'listen',
2797             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2798            {'call': 'copy_file_range',
2799             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2800            {'call': 'mq_timedsend',
2801             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2802            {'call': 'fgetxattr',
2803             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2804            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
2805            {'call': 'fcntl64',
2806             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2807            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
2808            {'call': 'fallocate',
2809             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2810            {'call': 'epoll_wait',
2811             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2812            {'call': 'llseek',
2813             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2814            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
2815            {'call': 'preadv64v2',
2816             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2817            {'call': 'readv',
2818             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2819            {'call': 'fstatfs',
2820             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2821            {'call': 'fstatfs64',
2822             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2823            {'call': 'write',
2824             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2825            {'call': 'mq_notify',
2826             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2827            {'call': 'sendfile',
2828             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2829            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
2830            {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
2831            {'call': 'bind', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
2832            {'call': 'flistxattr',
2833             'reason': set([('fd', 'file'), ('fd', 'flags')])},
2834            {'call': 'sendfile64',
2835             'reason': set([('fd', 'file'), ('fd', 'flags')])}],
2836  'fcntl64': [{'call': 'syncfs',
2837               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2838              {'call': 'vmsplice',
2839               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2840              {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
2841              {'call': 'pwritev64',
2842               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2843              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
2844              {'call': 'fremovexattr',
2845               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2846              {'call': 'readahead',
2847               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2848              {'call': 'getdents',
2849               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2850              {'call': 'writev',
2851               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2852              {'call': 'preadv64',
2853               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2854              {'call': 'fchmod',
2855               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2856              {'call': 'pread64',
2857               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2858              {'call': 'signalfd4',
2859               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2860              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
2861              {'call': 'remap_file_pages',
2862               'reason': set([('file', 'f_mode')])},
2863              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
2864              {'call': 'read',
2865               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2866              {'call': 'fchown',
2867               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2868              {'call': 'mq_timedreceive',
2869               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2870              {'call': 'utime',
2871               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2872              {'call': 'fsync',
2873               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2874              {'call': 'bpf',
2875               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2876              {'call': 'recvfrom',
2877               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2878              {'call': 'fsetxattr',
2879               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2880              {'call': 'sendto',
2881               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2882              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
2883              {'call': 'tee',
2884               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2885              {'call': 'sync_file_range',
2886               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2887              {'call': 'lseek',
2888               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2889              {'call': 'connect',
2890               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2891              {'call': 'getsockname',
2892               'reason': set([('fd', 'file'), ('fd', 'flags')])},
2893              {'call': 'epoll_ctl',
2894               'reason': set([('fd', 'file'),
2895                              ('fd', 'flags'),
2896                              ('file', 'f_mode')])},
2897              {'call': 'flock',
```

```
2898              'reason': set([('fd', 'file'),
2899                             ('fd', 'flags'),
2900                             ('file', 'f_mode')])},
2901             {'call': 'pwritev',
2902              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2903             {'call': 'fchdir',
2904              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2905             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
2906             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
2907             {'call': 'accept4',
2908              'reason': set([('fd', 'file'),
2909                             ('fd', 'flags'),
2910                             ('file', 'f_mode')])},
2911             {'call': 'old_readdir',
2912              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2913             {'call': 'inotify_rm_watch',
2914              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2915             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
2916             {'call': 'utimensat',
2917              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2918             {'call': 'inotify_add_watch',
2919              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2920             {'call': 'preadv2',
2921              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2922             {'call': 'splice',
2923              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2924             {'call': 'ftruncate',
2925              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2926             {'call': 'preadv',
2927              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2928             {'call': 'getpeername',
2929              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2930             {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
2931             {'call': 'setsockopt',
2932              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2933             {'call': 'socket', 'reason': set([('file', 'f_mode')])},
2934             {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
2935             {'call': 'fcntl',
2936              'reason': set([('fd', 'file'),
2937                             ('fd', 'flags'),
2938                             ('flock', 'l_len'),
2939                             ('flock', 'l_start')])},
2940             {'call': 'ioctl',
2941              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2942             {'call': 'pwrite64',
2943              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2944             {'call': 'perf_event_open',
2945              'reason': set([('fd', 'file'),
2946                             ('fd', 'flags'),
2947                             ('file', 'f_mode')])},
2948             {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
2949             {'call': 'pwritev64v2',
2950              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2951             {'call': 'futimesat',
2952              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2953             {'call': 'pwritev2',
2954              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2955             {'call': 'shutdown',
2956              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2957             {'call': 'acct', 'reason': set([('file', 'f_mode')])},
2958             {'call': 'open', 'reason': set([('file', 'f_mode')])},
2959             {'call': 'getsockopt',
2960              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2961             {'call': 'mq_getsetattr',
2962              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2963             {'call': 'dup', 'reason': set([('file', 'f_mode')])},
```



```
2964             {'call': 'fdatasync',
2965              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2966             {'call': 'setns', 'reason': set([('file', 'f_mode')])},
2967             {'call': 'getdents64',
2968              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2969             {'call': 'listen',
2970              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2971             {'call': 'copy_file_range',
2972              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2973             {'call': 'mq_timedsend',
2974              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2975             {'call': 'fgetxattr',
2976              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2977             {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
2978             {'call': 'fcntl64',
2979              'reason': set([('fd', 'file'),
2980                             ('fd', 'flags'),
2981                             ('flock', 'l_len'),
2982                             ('flock', 'l_start')])},
2983             {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
2984             {'call': 'fallocate',
2985              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2986             {'call': 'epoll_wait',
2987              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2988             {'call': 'llseek',
2989              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2990             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
2991             {'call': 'preadv64v2',
2992              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2993             {'call': 'readv',
2994              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2995             {'call': 'fstatfs',
2996              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2997             {'call': 'fstatfs64',
2998              'reason': set([('fd', 'file'), ('fd', 'flags')])},
2999             {'call': 'write',
3000              'reason': set([('fd', 'file'), ('fd', 'flags')])},
3001             {'call': 'mq_notify',
3002              'reason': set([('fd', 'file'), ('fd', 'flags')])},
3003             {'call': 'sendfile',
3004              'reason': set([('fd', 'file'), ('fd', 'flags')])},
3005             {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
3006             {'call': 'open_by_handle_at',
3007              'reason': set([('file', 'f_mode')])},
3008             {'call': 'bind',
3009              'reason': set([('fd', 'file'), ('fd', 'flags')])},
3010             {'call': 'flistxattr',
3011              'reason': set([('fd', 'file'), ('fd', 'flags')])},
3012             {'call': 'sendfile64',
3013              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
3014  'fdatasync': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
3015                {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
3016                {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
3017                {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
3018                {'call': 'readahead', 'reason': set([('fd', 'file')])},
3019                {'call': 'getdents', 'reason': set([('fd', 'file')])},
3020                {'call': 'writev', 'reason': set([('fd', 'file')])},
3021                {'call': 'preadv64', 'reason': set([('fd', 'file')])},
3022                {'call': 'fchmod', 'reason': set([('fd', 'file')])},
3023                {'call': 'pread64', 'reason': set([('fd', 'file')])},
3024                {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
3025                {'call': 'read', 'reason': set([('fd', 'file')])},
3026                {'call': 'fchown', 'reason': set([('fd', 'file')])},
3027                {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
3028                {'call': 'utime', 'reason': set([('fd', 'file')])},
3029                {'call': 'fsync', 'reason': set([('fd', 'file')])},
```

```
3030                {'call': 'bpf', 'reason': set([('fd', 'file')])},
3031                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
3032                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
3033                {'call': 'sendto', 'reason': set([('fd', 'file')])},
3034                {'call': 'tee', 'reason': set([('fd', 'file')])},
3035                {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
3036                {'call': 'lseek', 'reason': set([('fd', 'file')])},
3037                {'call': 'connect', 'reason': set([('fd', 'file')])},
3038                {'call': 'getsockname', 'reason': set([('fd', 'file')])},
3039                {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
3040                {'call': 'flock', 'reason': set([('fd', 'file')])},
3041                {'call': 'pwritev', 'reason': set([('fd', 'file')])},
3042                {'call': 'fchdir', 'reason': set([('fd', 'file')])},
3043                {'call': 'accept4', 'reason': set([('fd', 'file')])},
3044                {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
3045                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
3046                {'call': 'utimensat', 'reason': set([('fd', 'file')])},
3047                {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
3048                {'call': 'preadv2', 'reason': set([('fd', 'file')])},
3049                {'call': 'splice', 'reason': set([('fd', 'file')])},
3050                {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
3051                {'call': 'preadv', 'reason': set([('fd', 'file')])},
3052                {'call': 'getpeername', 'reason': set([('fd', 'file')])},
3053                {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
3054                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
3055                {'call': 'ioctl', 'reason': set([('fd', 'file')])},
3056                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
3057                {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
3058                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
3059                {'call': 'futimesat', 'reason': set([('fd', 'file')])},
3060                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
3061                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
3062                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
3063                {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
3064                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
3065                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
3066                {'call': 'listen', 'reason': set([('fd', 'file')])},
3067                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
3068                {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
3069                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
3070                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
3071                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
3072                {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
3073                {'call': 'llseek', 'reason': set([('fd', 'file')])},
3074                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
3075                {'call': 'readv', 'reason': set([('fd', 'file')])},
3076                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
3077                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
3078                {'call': 'write', 'reason': set([('fd', 'file')])},
3079                {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
3080                {'call': 'sendfile', 'reason': set([('fd', 'file')])},
3081                {'call': 'bind', 'reason': set([('fd', 'file')])},
3082                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
3083                {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
3084  'fgetxattr': [{'call': 'syncfs',
3085                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3086                {'call': 'vmsplice',
3087                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3088                {'call': 'pwritev64',
3089                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3090                {'call': 'fremovexattr',
3091                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3092                {'call': 'readahead',
3093                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3094                {'call': 'getdents',
3095                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
3096                {'call': 'writev',
3097                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3098                {'call': 'preadv64',
3099                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3100                {'call': 'fchmod',
3101                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3102                {'call': 'pread64',
3103                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3104                {'call': 'signalfd4',
3105                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3106                {'call': 'read',
3107                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3108                {'call': 'fchown',
3109                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3110                {'call': 'mq_timedreceive',
3111                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3112                {'call': 'utime',
3113                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3114                {'call': 'fsync',
3115                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3116                {'call': 'bpf',
3117                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3118                {'call': 'recvfrom',
3119                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3120                {'call': 'fsetxattr',
3121                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3122                {'call': 'sendto',
3123                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3124                {'call': 'tee',
3125                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3126                {'call': 'sync_file_range',
3127                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3128                {'call': 'lseek',
3129                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3130                {'call': 'connect',
3131                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3132                {'call': 'getsockname',
3133                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3134                {'call': 'epoll_ctl',
3135                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3136                {'call': 'flock',
3137                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3138                {'call': 'pwritev',
3139                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3140                {'call': 'fchdir',
3141                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3142                {'call': 'accept4',
3143                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3144                {'call': 'old_readdir',
3145                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3146                {'call': 'inotify_rm_watch',
3147                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3148                {'call': 'utimensat',
3149                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3150                {'call': 'inotify_add_watch',
3151                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3152                {'call': 'preadv2',
3153                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3154                {'call': 'splice',
3155                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3156                {'call': 'ftruncate',
3157                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3158                {'call': 'preadv',
3159                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3160                {'call': 'getpeername',
3161                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
3162                {'call': 'setsockopt',
3163                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3164                {'call': 'fcntl',
3165                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3166                {'call': 'ioctl',
3167                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3168                {'call': 'pwrite64',
3169                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3170                {'call': 'perf_event_open',
3171                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3172                {'call': 'pwritev64v2',
3173                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3174                {'call': 'futimesat',
3175                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3176                {'call': 'pwritev2',
3177                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3178                {'call': 'shutdown',
3179                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3180                {'call': 'getsockopt',
3181                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3182                {'call': 'mq_getsetattr',
3183                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3184                {'call': 'fdatasync',
3185                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3186                {'call': 'getdents64',
3187                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3188                {'call': 'listen',
3189                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3190                {'call': 'copy_file_range',
3191                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3192                {'call': 'mq_timedsend',
3193                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3194                {'call': 'fgetxattr',
3195                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3196                {'call': 'fcntl64',
3197                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3198                {'call': 'fallocate',
3199                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3200                {'call': 'epoll_wait',
3201                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3202                {'call': 'llseek',
3203                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3204                {'call': 'preadv64v2',
3205                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3206                {'call': 'readv',
3207                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3208                {'call': 'fstatfs',
3209                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3210                {'call': 'fstatfs64',
3211                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3212                {'call': 'write',
3213                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3214                {'call': 'mq_notify',
3215                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3216                {'call': 'sendfile',
3217                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3218                {'call': 'bind',
3219                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3220                {'call': 'flistxattr',
3221                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3222                {'call': 'sendfile64',
3223                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
3224  'finit_module': [{'call': 'delete_module',
3225                    'reason': set([('module', 'args'),
3226                                   ('module', 'kp'),
3227                                   ('module', 'num_kp'),
```



```
3228                                   ('module_layout', 'base'),
3229                                   ('module_layout', 'size')])},
3230                {'call': 'init_module',
3231                 'reason': set([('load_info', 'debug'),
3232                                ('load_info', 'num_debug'),
3233                                ('module', 'args'),
3234                                ('module', 'kp'),
3235                                ('module', 'num_kp'),
3236                                ('module_layout', 'base'),
3237                                ('module_layout', 'size')])},
3238                {'call': 'finit_module',
3239                 'reason': set([('load_info', 'debug'),
3240                                ('load_info', 'num_debug'),
3241                                ('module', 'args'),
3242                                ('module', 'kp'),
3243                                ('module', 'num_kp'),
3244                                ('module_layout', 'base'),
3245                                ('module_layout', 'size')])}],
3246  'flistxattr': [{'call': 'syncfs',
3247                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3248                 {'call': 'vmsplice',
3249                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3250                 {'call': 'pwritev64',
3251                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3252                 {'call': 'fremovexattr',
3253                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3254                 {'call': 'readahead',
3255                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3256                 {'call': 'getdents',
3257                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3258                 {'call': 'writev',
3259                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3260                 {'call': 'preadv64',
3261                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3262                 {'call': 'fchmod',
3263                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3264                 {'call': 'pread64',
3265                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3266                 {'call': 'signalfd4',
3267                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3268                 {'call': 'read',
3269                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3270                 {'call': 'fchown',
3271                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3272                 {'call': 'mq_timedreceive',
3273                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3274                 {'call': 'utime',
3275                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3276                 {'call': 'fsync',
3277                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3278                 {'call': 'bpf',
3279                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3280                 {'call': 'recvfrom',
3281                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3282                 {'call': 'fsetxattr',
3283                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3284                 {'call': 'sendto',
3285                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3286                 {'call': 'tee',
3287                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3288                 {'call': 'sync_file_range',
3289                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3290                 {'call': 'lseek',
3291                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3292                 {'call': 'connect',
3293                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
3294                 {'call': 'getsockname',
3295                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3296                 {'call': 'epoll_ctl',
3297                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3298                 {'call': 'flock',
3299                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3300                 {'call': 'pwritev',
3301                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3302                 {'call': 'fchdir',
3303                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3304                 {'call': 'accept4',
3305                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3306                 {'call': 'old_readdir',
3307                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3308                 {'call': 'inotify_rm_watch',
3309                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3310                 {'call': 'utimensat',
3311                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3312                 {'call': 'inotify_add_watch',
3313                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3314                 {'call': 'preadv2',
3315                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3316                 {'call': 'splice',
3317                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3318                 {'call': 'ftruncate',
3319                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3320                 {'call': 'preadv',
3321                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3322                 {'call': 'getpeername',
3323                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3324                 {'call': 'setsockopt',
3325                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3326                 {'call': 'fcntl',
3327                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3328                 {'call': 'ioctl',
3329                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3330                 {'call': 'pwrite64',
3331                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3332                 {'call': 'perf_event_open',
3333                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3334                 {'call': 'pwritev64v2',
3335                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3336                 {'call': 'futimesat',
3337                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3338                 {'call': 'pwritev2',
3339                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3340                 {'call': 'shutdown',
3341                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3342                 {'call': 'getsockopt',
3343                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3344                 {'call': 'mq_getsetattr',
3345                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3346                 {'call': 'fdatasync',
3347                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3348                 {'call': 'getdents64',
3349                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3350                 {'call': 'listen',
3351                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3352                 {'call': 'copy_file_range',
3353                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3354                 {'call': 'mq_timedsend',
3355                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3356                 {'call': 'fgetxattr',
3357                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3358                 {'call': 'fcntl64',
3359                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
3360                 {'call': 'fallocate',
3361                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3362                 {'call': 'epoll_wait',
3363                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3364                 {'call': 'llseek',
3365                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3366                 {'call': 'preadv64v2',
3367                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3368                 {'call': 'readv',
3369                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3370                 {'call': 'fstatfs',
3371                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3372                 {'call': 'fstatfs64',
3373                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3374                 {'call': 'write',
3375                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3376                 {'call': 'mq_notify',
3377                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3378                 {'call': 'sendfile',
3379                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3380                 {'call': 'bind',
3381                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3382                 {'call': 'flistxattr',
3383                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
3384                 {'call': 'sendfile64',
3385                  'reason': set([('fd', 'file'), ('fd', 'flags')])}],
3386  'flock': [{'call': 'syncfs',
3387             'reason': set([('fd', 'file'),
3388                            ('fd', 'flags'),
3389                            ('super_block', 's_flags')])},
3390            {'call': 'vmsplice',
3391             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3392            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
3393            {'call': 'pwritev64',
3394             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3395            {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
3396            {'call': 'fremovexattr',
3397             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3398            {'call': 'readahead',
3399             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3400            {'call': 'getdents',
3401             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3402            {'call': 'writev',
3403             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3404            {'call': 'preadv64',
3405             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3406            {'call': 'fchmod',
3407             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3408            {'call': 'pread64',
3409             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3410            {'call': 'signalfd4',
3411             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3412            {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
3413            {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
3414            {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
3415            {'call': 'read', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3416            {'call': 'fchown',
3417             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3418            {'call': 'mq_timedreceive',
3419             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3420            {'call': 'utime',
3421             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3422            {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
3423            {'call': 'fsync',
3424             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3425            {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
3426            {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
3427            {'call': 'recvfrom',
3428             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3429            {'call': 'fsetxattr',
3430             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3431            {'call': 'sendto',
3432             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3433            {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
3434            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3435            {'call': 'sync_file_range',
3436             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3437            {'call': 'lseek',
3438             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3439            {'call': 'connect',
3440             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3441            {'call': 'getsockname',
3442             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3443            {'call': 'epoll_ctl',
3444             'reason': set([('fd', 'file'),
3445                            ('fd', 'flags'),
3446                            ('file', 'f_mode')])},
3447            {'call': 'flock',
3448             'reason': set([('fd', 'file'),
3449                            ('fd', 'flags'),
3450                            ('file', 'f_mode')])},
3451            {'call': 'pwritev',
3452             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3453            {'call': 'fchdir',
3454             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3455            {'call': 'openat', 'reason': set([('file', 'f_mode')])},
3456            {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
3457            {'call': 'accept4',
3458             'reason': set([('fd', 'file'),
3459                            ('fd', 'flags'),
3460                            ('file', 'f_mode')])},
3461            {'call': 'old_readdir',
3462             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3463            {'call': 'inotify_rm_watch',
3464             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3465            {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
3466            {'call': 'utimensat',
3467             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3468            {'call': 'inotify_add_watch',
3469             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3470            {'call': 'preadv2',
3471             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3472            {'call': 'splice',
3473             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3474            {'call': 'ftruncate',
3475             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3476            {'call': 'preadv',
3477             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3478            {'call': 'getpeername',
3479             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3480            {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
3481            {'call': 'setsockopt',
3482             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3483            {'call': 'socket', 'reason': set([('file', 'f_mode')])},
3484            {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
3485            {'call': 'fcntl',
3486             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3487            {'call': 'ioctl',
3488             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3489            {'call': 'pwrite64',
3490             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3491            {'call': 'perf_event_open',
```



```
3492             'reason': set([('fd', 'file'),
3493                            ('fd', 'flags'),
3494                            ('file', 'f_mode')])},
3495            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
3496            {'call': 'pwritev64v2',
3497             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3498            {'call': 'quotactl', 'reason': set([('super_block', 's_flags')])},
3499            {'call': 'futimesat',
3500             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3501            {'call': 'pwritev2',
3502             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3503            {'call': 'shutdown',
3504             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3505            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
3506            {'call': 'open', 'reason': set([('file', 'f_mode')])},
3507            {'call': 'getsockopt',
3508             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3509            {'call': 'mq_getsetattr',
3510             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3511            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
3512            {'call': 'fdatasync',
3513             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3514            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
3515            {'call': 'getdents64',
3516             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3517            {'call': 'listen',
3518             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3519            {'call': 'copy_file_range',
3520             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3521            {'call': 'mq_timedsend',
3522             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3523            {'call': 'fgetxattr',
3524             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3525            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
3526            {'call': 'fcntl64',
3527             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3528            {'call': 'swapon',
3529             'reason': set([('file', 'f_mode'), ('super_block', 's_flags')])},
3530            {'call': 'fallocate',
3531             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3532            {'call': 'epoll_wait',
3533             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3534            {'call': 'llseek',
3535             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3536            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
3537            {'call': 'preadv64v2',
3538             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3539            {'call': 'readv',
3540             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3541            {'call': 'fstatfs',
3542             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3543            {'call': 'fstatfs64',
3544             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3545            {'call': 'write',
3546             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3547            {'call': 'mq_notify',
3548             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3549            {'call': 'sendfile',
3550             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3551            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
3552            {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
3553            {'call': 'bind', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3554            {'call': 'flistxattr',
3555             'reason': set([('fd', 'file'), ('fd', 'flags')])},
3556            {'call': 'sendfile64',
3557             'reason': set([('fd', 'file'), ('fd', 'flags')])}],
```

```
3558  'fremovexattr': [{'call': 'syncfs',
3559                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3560                   {'call': 'vmsplice',
3561                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3562                   {'call': 'pwritev64',
3563                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3564                   {'call': 'fremovexattr',
3565                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3566                   {'call': 'readahead',
3567                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3568                   {'call': 'getdents',
3569                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3570                   {'call': 'writev',
3571                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3572                   {'call': 'preadv64',
3573                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3574                   {'call': 'fchmod',
3575                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3576                   {'call': 'pread64',
3577                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3578                   {'call': 'signalfd4',
3579                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3580                   {'call': 'read',
3581                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3582                   {'call': 'fchown',
3583                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3584                   {'call': 'mq_timedreceive',
3585                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3586                   {'call': 'utime',
3587                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3588                   {'call': 'fsync',
3589                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3590                   {'call': 'bpf',
3591                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3592                   {'call': 'recvfrom',
3593                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3594                   {'call': 'fsetxattr',
3595                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3596                   {'call': 'sendto',
3597                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3598                   {'call': 'tee',
3599                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3600                   {'call': 'sync_file_range',
3601                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3602                   {'call': 'lseek',
3603                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3604                   {'call': 'connect',
3605                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3606                   {'call': 'getsockname',
3607                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3608                   {'call': 'epoll_ctl',
3609                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3610                   {'call': 'flock',
3611                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3612                   {'call': 'pwritev',
3613                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3614                   {'call': 'fchdir',
3615                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3616                   {'call': 'accept4',
3617                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3618                   {'call': 'old_readdir',
3619                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3620                   {'call': 'inotify_rm_watch',
3621                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3622                   {'call': 'utimensat',
3623                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
3624                   {'call': 'inotify_add_watch',
3625                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3626                   {'call': 'preadv2',
3627                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3628                   {'call': 'splice',
3629                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3630                   {'call': 'ftruncate',
3631                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3632                   {'call': 'preadv',
3633                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3634                   {'call': 'getpeername',
3635                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3636                   {'call': 'setsockopt',
3637                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3638                   {'call': 'fcntl',
3639                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3640                   {'call': 'ioctl',
3641                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3642                   {'call': 'pwrite64',
3643                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3644                   {'call': 'perf_event_open',
3645                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3646                   {'call': 'pwritev64v2',
3647                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3648                   {'call': 'futimesat',
3649                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3650                   {'call': 'pwritev2',
3651                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3652                   {'call': 'shutdown',
3653                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3654                   {'call': 'getsockopt',
3655                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3656                   {'call': 'mq_getsetattr',
3657                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3658                   {'call': 'fdatasync',
3659                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3660                   {'call': 'getdents64',
3661                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3662                   {'call': 'listen',
3663                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3664                   {'call': 'copy_file_range',
3665                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3666                   {'call': 'mq_timedsend',
3667                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3668                   {'call': 'fgetxattr',
3669                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3670                   {'call': 'fcntl64',
3671                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3672                   {'call': 'fallocate',
3673                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3674                   {'call': 'epoll_wait',
3675                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3676                   {'call': 'llseek',
3677                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3678                   {'call': 'preadv64v2',
3679                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3680                   {'call': 'readv',
3681                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3682                   {'call': 'fstatfs',
3683                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3684                   {'call': 'fstatfs64',
3685                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3686                   {'call': 'write',
3687                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
3688                   {'call': 'mq_notify',
3689                    'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
                {'call': 'sendfile',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'bind',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'flistxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sendfile64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
 'fsetxattr': [{'call': 'syncfs',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'vmsplice',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fremovexattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'readahead',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getdents',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'writev',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fchmod',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pread64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'signalfd4',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'read',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fchown',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'mq_timedreceive',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'utime',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fsync',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'bpf',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'recvfrom',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fsetxattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'sendto',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'tee',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'sync_file_range',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'lseek',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'connect',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getsockname',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'epoll_ctl',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'flock',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fchdir',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
               {'call': 'accept4',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'old_readdir',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'inotify_rm_watch',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'utimensat',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'inotify_add_watch',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'splice',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'ftruncate',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getpeername',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'setsockopt',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fcntl',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'ioctl',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwrite64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'perf_event_open',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev64v2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'futimesat',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'shutdown',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getsockopt',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'mq_getsetattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fdatasync',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'getdents64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'listen',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'copy_file_range',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'mq_timedsend',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fgetxattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fcntl64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fallocate',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'epoll_wait',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'llseek',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'preadv64v2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'readv',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
3822                {'call': 'fstatfs',
3823                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3824                {'call': 'fstatfs64',
3825                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3826                {'call': 'write',
3827                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3828                {'call': 'mq_notify',
3829                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3830                {'call': 'sendfile',
3831                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3832                {'call': 'bind',
3833                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3834                {'call': 'flistxattr',
3835                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
3836                {'call': 'sendfile64',
3837                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
3838  'fstat': [{'call': 'lstat',
3839             'reason': set([('__old_kernel_stat', 'st_ino'),
3840                            ('__old_kernel_stat', 'st_nlink')])},
3841            {'call': 'stat',
3842             'reason': set([('__old_kernel_stat', 'st_ino'),
3843                            ('__old_kernel_stat', 'st_nlink')])},
3844            {'call': 'fstat',
3845             'reason': set([('__old_kernel_stat', 'st_ino'),
3846                            ('__old_kernel_stat', 'st_nlink'),
3847                            ('kstat', 'dev'),
3848                            ('kstat', 'ino'),
3849                            ('kstat', 'nlink'),
3850                            ('kstat', 'rdev')])},
3851            {'call': 'newfstat',
3852             'reason': set([('kstat', 'dev'),
3853                            ('kstat', 'ino'),
3854                            ('kstat', 'nlink'),
3855                            ('kstat', 'rdev')])}],
3856  'fstatfs': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
3857              {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
3858              {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
3859              {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
3860              {'call': 'readahead', 'reason': set([('fd', 'file')])},
3861              {'call': 'getdents', 'reason': set([('fd', 'file')])},
3862              {'call': 'writev', 'reason': set([('fd', 'file')])},
3863              {'call': 'preadv64', 'reason': set([('fd', 'file')])},
3864              {'call': 'fchmod', 'reason': set([('fd', 'file')])},
3865              {'call': 'pread64', 'reason': set([('fd', 'file')])},
3866              {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
3867              {'call': 'read', 'reason': set([('fd', 'file')])},
3868              {'call': 'fchown', 'reason': set([('fd', 'file')])},
3869              {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
3870              {'call': 'utime', 'reason': set([('fd', 'file')])},
3871              {'call': 'ustat',
3872               'reason': set([('kstatfs', 'f_ffree'),
3873                              ('kstatfs', 'f_files')])},
3874              {'call': 'fsync', 'reason': set([('fd', 'file')])},
3875              {'call': 'bpf', 'reason': set([('fd', 'file')])},
3876              {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
3877              {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
3878              {'call': 'sendto', 'reason': set([('fd', 'file')])},
3879              {'call': 'tee', 'reason': set([('fd', 'file')])},
3880              {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
3881              {'call': 'lseek', 'reason': set([('fd', 'file')])},
3882              {'call': 'connect', 'reason': set([('fd', 'file')])},
3883              {'call': 'getsockname', 'reason': set([('fd', 'file')])},
3884              {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
3885              {'call': 'flock', 'reason': set([('fd', 'file')])},
3886              {'call': 'pwritev', 'reason': set([('fd', 'file')])},
3887              {'call': 'fchdir', 'reason': set([('fd', 'file')])},
```



```
3888              {'call': 'accept4', 'reason': set([('fd', 'file')])},
3889              {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
3890              {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
3891              {'call': 'utimensat', 'reason': set([('fd', 'file')])},
3892              {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
3893              {'call': 'preadv2', 'reason': set([('fd', 'file')])},
3894              {'call': 'splice', 'reason': set([('fd', 'file')])},
3895              {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
3896              {'call': 'preadv', 'reason': set([('fd', 'file')])},
3897              {'call': 'getpeername', 'reason': set([('fd', 'file')])},
3898              {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
3899              {'call': 'fcntl', 'reason': set([('fd', 'file')])},
3900              {'call': 'ioctl', 'reason': set([('fd', 'file')])},
3901              {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
3902              {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
3903              {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
3904              {'call': 'futimesat', 'reason': set([('fd', 'file')])},
3905              {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
3906              {'call': 'shutdown', 'reason': set([('fd', 'file')])},
3907              {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
3908              {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
3909              {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
3910              {'call': 'getdents64', 'reason': set([('fd', 'file')])},
3911              {'call': 'listen', 'reason': set([('fd', 'file')])},
3912              {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
3913              {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
3914              {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
3915              {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
3916              {'call': 'fallocate', 'reason': set([('fd', 'file')])},
3917              {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
3918              {'call': 'llseek', 'reason': set([('fd', 'file')])},
3919              {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
3920              {'call': 'readv', 'reason': set([('fd', 'file')])},
3921              {'call': 'fstatfs',
3922               'reason': set([('fd', 'file'),
3923                              ('kstatfs', 'f_ffree'),
3924                              ('kstatfs', 'f_files')])},
3925              {'call': 'statfs',
3926               'reason': set([('kstatfs', 'f_ffree'),
3927                              ('kstatfs', 'f_files')])},
3928              {'call': 'fstatfs64',
3929               'reason': set([('fd', 'file'),
3930                              ('kstatfs', 'f_ffree'),
3931                              ('kstatfs', 'f_files')])},
3932              {'call': 'write', 'reason': set([('fd', 'file')])},
3933              {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
3934              {'call': 'sendfile', 'reason': set([('fd', 'file')])},
3935              {'call': 'statfs64',
3936               'reason': set([('kstatfs', 'f_ffree'),
3937                              ('kstatfs', 'f_files')])},
3938              {'call': 'bind', 'reason': set([('fd', 'file')])},
3939              {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
3940              {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
3941  'fstatfs64': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
3942                {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
3943                {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
3944                {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
3945                {'call': 'readahead', 'reason': set([('fd', 'file')])},
3946                {'call': 'getdents', 'reason': set([('fd', 'file')])},
3947                {'call': 'writev', 'reason': set([('fd', 'file')])},
3948                {'call': 'preadv64', 'reason': set([('fd', 'file')])},
3949                {'call': 'fchmod', 'reason': set([('fd', 'file')])},
3950                {'call': 'pread64', 'reason': set([('fd', 'file')])},
3951                {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
3952                {'call': 'read', 'reason': set([('fd', 'file')])},
3953                {'call': 'fchown', 'reason': set([('fd', 'file')])},
```

```
3954                {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
3955                {'call': 'utime', 'reason': set([('fd', 'file')])},
3956                {'call': 'ustat',
3957                 'reason': set([('kstatfs', 'f_ffree'),
3958                                ('kstatfs', 'f_files')])},
3959                {'call': 'fsync', 'reason': set([('fd', 'file')])},
3960                {'call': 'bpf', 'reason': set([('fd', 'file')])},
3961                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
3962                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
3963                {'call': 'sendto', 'reason': set([('fd', 'file')])},
3964                {'call': 'tee', 'reason': set([('fd', 'file')])},
3965                {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
3966                {'call': 'lseek', 'reason': set([('fd', 'file')])},
3967                {'call': 'connect', 'reason': set([('fd', 'file')])},
3968                {'call': 'getsockname', 'reason': set([('fd', 'file')])},
3969                {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
3970                {'call': 'flock', 'reason': set([('fd', 'file')])},
3971                {'call': 'pwritev', 'reason': set([('fd', 'file')])},
3972                {'call': 'fchdir', 'reason': set([('fd', 'file')])},
3973                {'call': 'accept4', 'reason': set([('fd', 'file')])},
3974                {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
3975                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
3976                {'call': 'utimensat', 'reason': set([('fd', 'file')])},
3977                {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
3978                {'call': 'preadv2', 'reason': set([('fd', 'file')])},
3979                {'call': 'splice', 'reason': set([('fd', 'file')])},
3980                {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
3981                {'call': 'preadv', 'reason': set([('fd', 'file')])},
3982                {'call': 'getpeername', 'reason': set([('fd', 'file')])},
3983                {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
3984                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
3985                {'call': 'ioctl', 'reason': set([('fd', 'file')])},
3986                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
3987                {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
3988                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
3989                {'call': 'futimesat', 'reason': set([('fd', 'file')])},
3990                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
3991                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
3992                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
3993                {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
3994                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
3995                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
3996                {'call': 'listen', 'reason': set([('fd', 'file')])},
3997                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
3998                {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
3999                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
4000                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
4001                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
4002                {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
4003                {'call': 'llseek', 'reason': set([('fd', 'file')])},
4004                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
4005                {'call': 'readv', 'reason': set([('fd', 'file')])},
4006                {'call': 'fstatfs',
4007                 'reason': set([('fd', 'file'),
4008                                ('kstatfs', 'f_ffree'),
4009                                ('kstatfs', 'f_files')])},
4010                {'call': 'statfs',
4011                 'reason': set([('kstatfs', 'f_ffree'),
4012                                ('kstatfs', 'f_files')])},
4013                {'call': 'fstatfs64',
4014                 'reason': set([('fd', 'file'),
4015                                ('kstatfs', 'f_ffree'),
4016                                ('kstatfs', 'f_files')])},
4017                {'call': 'write', 'reason': set([('fd', 'file')])},
4018                {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
4019                {'call': 'sendfile', 'reason': set([('fd', 'file')])},
```



```
4020                {'call': 'statfs64',
4021                 'reason': set([('kstatfs', 'f_ffree'),
4022                                ('kstatfs', 'f_files')])},
4023                {'call': 'bind', 'reason': set([('fd', 'file')])},
4024                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
4025                {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
4026  'fsync': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
4027            {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
4028            {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
4029            {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
4030            {'call': 'readahead', 'reason': set([('fd', 'file')])},
4031            {'call': 'getdents', 'reason': set([('fd', 'file')])},
4032            {'call': 'writev', 'reason': set([('fd', 'file')])},
4033            {'call': 'preadv64', 'reason': set([('fd', 'file')])},
4034            {'call': 'fchmod', 'reason': set([('fd', 'file')])},
4035            {'call': 'pread64', 'reason': set([('fd', 'file')])},
4036            {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
4037            {'call': 'read', 'reason': set([('fd', 'file')])},
4038            {'call': 'fchown', 'reason': set([('fd', 'file')])},
4039            {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
4040            {'call': 'utime', 'reason': set([('fd', 'file')])},
4041            {'call': 'fsync', 'reason': set([('fd', 'file')])},
4042            {'call': 'bpf', 'reason': set([('fd', 'file')])},
4043            {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
4044            {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
4045            {'call': 'sendto', 'reason': set([('fd', 'file')])},
4046            {'call': 'tee', 'reason': set([('fd', 'file')])},
4047            {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
4048            {'call': 'lseek', 'reason': set([('fd', 'file')])},
4049            {'call': 'connect', 'reason': set([('fd', 'file')])},
4050            {'call': 'getsockname', 'reason': set([('fd', 'file')])},
4051            {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
4052            {'call': 'flock', 'reason': set([('fd', 'file')])},
4053            {'call': 'pwritev', 'reason': set([('fd', 'file')])},
4054            {'call': 'fchdir', 'reason': set([('fd', 'file')])},
4055            {'call': 'accept4', 'reason': set([('fd', 'file')])},
4056            {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
4057            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
4058            {'call': 'utimensat', 'reason': set([('fd', 'file')])},
4059            {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
4060            {'call': 'preadv2', 'reason': set([('fd', 'file')])},
4061            {'call': 'splice', 'reason': set([('fd', 'file')])},
4062            {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
4063            {'call': 'preadv', 'reason': set([('fd', 'file')])},
4064            {'call': 'getpeername', 'reason': set([('fd', 'file')])},
4065            {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
4066            {'call': 'fcntl', 'reason': set([('fd', 'file')])},
4067            {'call': 'ioctl', 'reason': set([('fd', 'file')])},
4068            {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
4069            {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
4070            {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
4071            {'call': 'futimesat', 'reason': set([('fd', 'file')])},
4072            {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
4073            {'call': 'shutdown', 'reason': set([('fd', 'file')])},
4074            {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
4075            {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
4076            {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
4077            {'call': 'getdents64', 'reason': set([('fd', 'file')])},
4078            {'call': 'listen', 'reason': set([('fd', 'file')])},
4079            {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
4080            {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
4081            {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
4082            {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
4083            {'call': 'fallocate', 'reason': set([('fd', 'file')])},
4084            {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
4085            {'call': 'llseek', 'reason': set([('fd', 'file')])},
```

```
4086            {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
4087            {'call': 'readv', 'reason': set([('fd', 'file')])},
4088            {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
4089            {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
4090            {'call': 'write', 'reason': set([('fd', 'file')])},
4091            {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
4092            {'call': 'sendfile', 'reason': set([('fd', 'file')])},
4093            {'call': 'bind', 'reason': set([('fd', 'file')])},
4094            {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
4095            {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
4096  'ftruncate': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
4097                {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
4098                {'call': 'eventfd2',
4099                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4100                {'call': 'mq_unlink',
4101                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4102                {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
4103                {'call': 'swapoff',
4104                 'reason': set([('file', 'f_flags'),
4105                                ('file', 'f_mode'),
4106                                ('inode', 'i_flags'),
4107                                ('inode', 'i_sb')])},
4108                {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
4109                {'call': 'readahead', 'reason': set([('fd', 'file')])},
4110                {'call': 'getdents', 'reason': set([('fd', 'file')])},
4111                {'call': 'writev', 'reason': set([('fd', 'file')])},
4112                {'call': 'preadv64', 'reason': set([('fd', 'file')])},
4113                {'call': 'fchmod',
4114                 'reason': set([('fd', 'file'),
4115                                ('inode', 'i_flags'),
4116                                ('inode', 'i_sb')])},
4117                {'call': 'pread64', 'reason': set([('fd', 'file')])},
4118                {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
4119                {'call': 'memfd_create',
4120                 'reason': set([('file', 'f_flags'),
4121                                ('file', 'f_mode'),
4122                                ('inode', 'i_flags'),
4123                                ('inode', 'i_sb')])},
4124                {'call': 'remap_file_pages',
4125                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4126                {'call': 'dup3',
4127                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4128                {'call': 'readlinkat',
4129                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4130                {'call': 'read', 'reason': set([('fd', 'file')])},
4131                {'call': 'fchown',
4132                 'reason': set([('fd', 'file'),
4133                                ('inode', 'i_flags'),
4134                                ('inode', 'i_sb')])},
4135                {'call': 'mq_timedreceive',
4136                 'reason': set([('fd', 'file'),
4137                                ('inode', 'i_flags'),
4138                                ('inode', 'i_sb')])},
4139                {'call': 'utime', 'reason': set([('fd', 'file')])},
4140                {'call': 'fsync', 'reason': set([('fd', 'file')])},
4141                {'call': 'bpf', 'reason': set([('fd', 'file')])},
4142                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
4143                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
4144                {'call': 'sendto', 'reason': set([('fd', 'file')])},
4145                {'call': 'epoll_create1',
4146                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4147                {'call': 'tee', 'reason': set([('fd', 'file')])},
4148                {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
4149                {'call': 'lseek', 'reason': set([('fd', 'file')])},
4150                {'call': 'connect', 'reason': set([('fd', 'file')])},
4151                {'call': 'getsockname', 'reason': set([('fd', 'file')])},
```



```
4152                {'call': 'epoll_ctl',
4153                 'reason': set([('fd', 'file'),
4154                                ('file', 'f_flags'),
4155                                ('file', 'f_mode')])},
4156                {'call': 'flock',
4157                 'reason': set([('fd', 'file'),
4158                                ('file', 'f_flags'),
4159                                ('file', 'f_mode')])},
4160                {'call': 'pwritev', 'reason': set([('fd', 'file')])},
4161                {'call': 'fchdir', 'reason': set([('fd', 'file')])},
4162                {'call': 'openat',
4163                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4164                {'call': 'uselib',
4165                 'reason': set([('file', 'f_flags'),
4166                                ('file', 'f_mode'),
4167                                ('inode', 'i_flags'),
4168                                ('inode', 'i_sb')])},
4169                {'call': 'accept4',
4170                 'reason': set([('fd', 'file'),
4171                                ('file', 'f_flags'),
4172                                ('file', 'f_mode')])},
4173                {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
4174                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
4175                {'call': 'socketpair',
4176                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4177                {'call': 'utimensat', 'reason': set([('fd', 'file')])},
4178                {'call': 'fchmodat',
4179                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4180                {'call': 'inotify_add_watch',
4181                 'reason': set([('fd', 'file'),
4182                                ('inode', 'i_flags'),
4183                                ('inode', 'i_sb')])},
4184                {'call': 'preadv2', 'reason': set([('fd', 'file')])},
4185                {'call': 'splice', 'reason': set([('fd', 'file')])},
4186                {'call': 'ftruncate',
4187                 'reason': set([('fd', 'file'),
4188                                ('inode', 'i_flags'),
4189                                ('inode', 'i_sb')])},
4190                {'call': 'preadv', 'reason': set([('fd', 'file')])},
4191                {'call': 'getpeername', 'reason': set([('fd', 'file')])},
4192                {'call': 'shmat',
4193                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4194                {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
4195                {'call': 'socket',
4196                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4197                {'call': 'pipe2',
4198                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4199                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
4200                {'call': 'ioctl',
4201                 'reason': set([('fd', 'file'),
4202                                ('inode', 'i_flags'),
4203                                ('inode', 'i_sb')])},
4204                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
4205                {'call': 'perf_event_open',
4206                 'reason': set([('fd', 'file'),
4207                                ('file', 'f_flags'),
4208                                ('file', 'f_mode')])},
4209                {'call': 'linkat',
4210                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4211                {'call': 'shmdt',
4212                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4213                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
4214                {'call': 'futimesat', 'reason': set([('fd', 'file')])},
4215                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
4216                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
4217                {'call': 'acct',
```

```
4218                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4219                {'call': 'open',
4220                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4221                {'call': 'unlink',
4222                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4223                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
4224                {'call': 'mq_getsetattr',
4225                 'reason': set([('fd', 'file'),
4226                                ('file', 'f_flags'),
4227                                ('inode', 'i_flags'),
4228                                ('inode', 'i_sb')])},
4229                {'call': 'faccessat',
4230                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4231                {'call': 'dup',
4232                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4233                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
4234                {'call': 'setns',
4235                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4236                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
4237                {'call': 'listen', 'reason': set([('fd', 'file')])},
4238                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
4239                {'call': 'mq_timedsend',
4240                 'reason': set([('fd', 'file'),
4241                                ('inode', 'i_flags'),
4242                                ('inode', 'i_sb')])},
4243                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
4244                {'call': 'shmctl',
4245                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4246                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
4247                {'call': 'swapon',
4248                 'reason': set([('file', 'f_flags'),
4249                                ('file', 'f_mode'),
4250                                ('inode', 'i_flags'),
4251                                ('inode', 'i_sb')])},
4252                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
4253                {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
4254                {'call': 'fchownat',
4255                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4256                {'call': 'llseek', 'reason': set([('fd', 'file')])},
4257                {'call': 'mmap_pgoff',
4258                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4259                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
4260                {'call': 'readv', 'reason': set([('fd', 'file')])},
4261                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
4262                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
4263                {'call': 'write', 'reason': set([('fd', 'file')])},
4264                {'call': 'mq_notify',
4265                 'reason': set([('fd', 'file'),
4266                                ('inode', 'i_flags'),
4267                                ('inode', 'i_sb')])},
4268                {'call': 'sendfile',
4269                 'reason': set([('fd', 'file'),
4270                                ('inode', 'i_flags'),
4271                                ('inode', 'i_sb')])},
4272                {'call': 'mq_open',
4273                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4274                {'call': 'unlinkat',
4275                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
4276                {'call': 'open_by_handle_at',
4277                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4278                {'call': 'bind', 'reason': set([('fd', 'file')])},
4279                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
4280                {'call': 'sendfile64',
4281                 'reason': set([('fd', 'file'),
4282                                ('inode', 'i_flags'),
4283                                ('inode', 'i_sb')])}],
```



```
4284  'futex': [{'call': 'rt_sigtimedwait',
4285             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4286            {'call': 'mq_unlink',
4287             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4288            {'call': 'swapoff',
4289             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4290            {'call': 'fchmod',
4291             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4292            {'call': 'memfd_create',
4293             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4294            {'call': 'readlinkat',
4295             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4296            {'call': 'io_getevents',
4297             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4298            {'call': 'fchown',
4299             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4300            {'call': 'mq_timedreceive',
4301             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4302            {'call': 'utime',
4303             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4304            {'call': 'semtimedop',
4305             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4306            {'call': 'settimeofday',
4307             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4308            {'call': 'sched_rr_get_interval',
4309             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4310            {'call': 'timerfd_gettime',
4311             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4312            {'call': 'pselect6',
4313             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4314            {'call': 'uselib',
4315             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4316            {'call': 'fchmodat',
4317             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4318            {'call': 'inotify_add_watch',
4319             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4320            {'call': 'timer_settime',
4321             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4322            {'call': 'ftruncate',
4323             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4324            {'call': 'timer_gettime',
4325             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4326            {'call': 'ioctl',
4327             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4328            {'call': 'linkat',
4329             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4330            {'call': 'stime',
4331             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4332            {'call': 'futimesat',
4333             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4334            {'call': 'poll',
4335             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4336            {'call': 'select',
4337             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4338            {'call': 'unlink',
4339             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4340            {'call': 'nanosleep',
4341             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4342            {'call': 'mq_getsetattr',
4343             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4344            {'call': 'faccessat',
4345             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4346            {'call': 'mq_timedsend',
4347             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4348            {'call': 'swapon',
4349             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
```

```
4350            {'call': 'epoll_wait',
4351             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4352            {'call': 'fchownat',
4353             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4354            {'call': 'fstat',
4355             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4356            {'call': 'timerfd_settime',
4357             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4358            {'call': 'mq_notify',
4359             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4360            {'call': 'sendfile',
4361             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4362            {'call': 'newfstat',
4363             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4364            {'call': 'clock_nanosleep',
4365             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4366            {'call': 'unlinkat',
4367             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4368            {'call': 'futex',
4369             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4370            {'call': 'recvmmsg',
4371             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4372            {'call': 'sendfile64',
4373             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
4374            {'call': 'ppoll',
4375             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])}],
4376  'futimesat': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
4377                {'call': 'rt_sigtimedwait',
4378                 'reason': set([('timespec', 'tv_nsec')])},
4379                {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
4380                {'call': 'mq_unlink',
4381                 'reason': set([('timespec', 'tv_nsec')])},
4382                {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
4383                {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
4384                {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
4385                {'call': 'readahead', 'reason': set([('fd', 'file')])},
4386                {'call': 'getdents', 'reason': set([('fd', 'file')])},
4387                {'call': 'writev', 'reason': set([('fd', 'file')])},
4388                {'call': 'preadv64', 'reason': set([('fd', 'file')])},
4389                {'call': 'fchmod',
4390                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4391                {'call': 'pread64', 'reason': set([('fd', 'file')])},
4392                {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
4393                {'call': 'memfd_create',
4394                 'reason': set([('timespec', 'tv_nsec')])},
4395                {'call': 'readlinkat',
4396                 'reason': set([('timespec', 'tv_nsec')])},
4397                {'call': 'read', 'reason': set([('fd', 'file')])},
4398                {'call': 'io_getevents',
4399                 'reason': set([('timespec', 'tv_nsec')])},
4400                {'call': 'fchown',
4401                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4402                {'call': 'waitid', 'reason': set([('timeval', 'tv_usec')])},
4403                {'call': 'mq_timedreceive',
4404                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4405                {'call': 'utime',
4406                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4407                {'call': 'fsync', 'reason': set([('fd', 'file')])},
4408                {'call': 'bpf', 'reason': set([('fd', 'file')])},
4409                {'call': 'semtimedop',
4410                 'reason': set([('timespec', 'tv_nsec')])},
4411                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
4412                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
4413                {'call': 'settimeofday',
4414                 'reason': set([('timespec', 'tv_nsec'),
4415                                ('timeval', 'tv_usec')])},
```



```
4416                {'call': 'sendto', 'reason': set([('fd', 'file')])},
4417                {'call': 'sched_rr_get_interval',
4418                 'reason': set([('timespec', 'tv_nsec')])},
4419                {'call': 'timerfd_gettime',
4420                 'reason': set([('timespec', 'tv_nsec')])},
4421                {'call': 'adjtimex', 'reason': set([('timeval', 'tv_usec')])},
4422                {'call': 'tee', 'reason': set([('fd', 'file')])},
4423                {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
4424                {'call': 'lseek', 'reason': set([('fd', 'file')])},
4425                {'call': 'connect', 'reason': set([('fd', 'file')])},
4426                {'call': 'getsockname', 'reason': set([('fd', 'file')])},
4427                {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
4428                {'call': 'flock', 'reason': set([('fd', 'file')])},
4429                {'call': 'pwritev', 'reason': set([('fd', 'file')])},
4430                {'call': 'fchdir', 'reason': set([('fd', 'file')])},
4431                {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
4432                {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
4433                {'call': 'accept4', 'reason': set([('fd', 'file')])},
4434                {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
4435                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
4436                {'call': 'utimensat', 'reason': set([('fd', 'file')])},
4437                {'call': 'getitimer', 'reason': set([('timeval', 'tv_usec')])},
4438                {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
4439                {'call': 'inotify_add_watch',
4440                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4441                {'call': 'preadv2', 'reason': set([('fd', 'file')])},
4442                {'call': 'timer_settime',
4443                 'reason': set([('timespec', 'tv_nsec')])},
4444                {'call': 'splice', 'reason': set([('fd', 'file')])},
4445                {'call': 'ftruncate',
4446                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4447                {'call': 'timer_gettime',
4448                 'reason': set([('timespec', 'tv_nsec')])},
4449                {'call': 'preadv', 'reason': set([('fd', 'file')])},
4450                {'call': 'getpeername', 'reason': set([('fd', 'file')])},
4451                {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
4452                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
4453                {'call': 'ioctl',
4454                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4455                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
4456                {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
4457                {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
4458                {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
4459                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
4460                {'call': 'futimesat',
4461                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4462                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
4463                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
4464                {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
4465                {'call': 'select',
4466                 'reason': set([('timespec', 'tv_nsec'),
4467                                ('timeval', 'tv_usec')])},
4468                {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
4469                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
4470                {'call': 'nanosleep',
4471                 'reason': set([('timespec', 'tv_nsec')])},
4472                {'call': 'mq_getsetattr',
4473                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4474                {'call': 'faccessat',
4475                 'reason': set([('timespec', 'tv_nsec')])},
4476                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
4477                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
4478                {'call': 'listen', 'reason': set([('fd', 'file')])},
4479                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
4480                {'call': 'mq_timedsend',
4481                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
```

```
4482                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
4483                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
4484                {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
4485                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
4486                {'call': 'wait4', 'reason': set([('timeval', 'tv_usec')])},
4487                {'call': 'epoll_wait',
4488                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4489                {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
4490                {'call': 'getrusage', 'reason': set([('timeval', 'tv_usec')])},
4491                {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
4492                {'call': 'timerfd_settime',
4493                 'reason': set([('timespec', 'tv_nsec')])},
4494                {'call': 'setitimer', 'reason': set([('timeval', 'tv_usec')])},
4495                {'call': 'llseek', 'reason': set([('fd', 'file')])},
4496                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
4497                {'call': 'readv', 'reason': set([('fd', 'file')])},
4498                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
4499                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
4500                {'call': 'write', 'reason': set([('fd', 'file')])},
4501                {'call': 'mq_notify',
4502                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4503                {'call': 'sendfile',
4504                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4505                {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
4506                {'call': 'clock_nanosleep',
4507                 'reason': set([('timespec', 'tv_nsec')])},
4508                {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
4509                {'call': 'clock_adjtime',
4510                 'reason': set([('timeval', 'tv_usec')])},
4511                {'call': 'bind', 'reason': set([('fd', 'file')])},
4512                {'call': 'alarm', 'reason': set([('timeval', 'tv_usec')])},
4513                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
4514                {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
4515                {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
4516                {'call': 'sendfile64',
4517                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
4518                {'call': 'ppoll',
4519                 'reason': set([('timespec', 'tv_nsec'),
4520                                ('timeval', 'tv_usec')])}],
4521  'get_mempolicy': [{'call': 'keyctl',
4522                     'reason': set([('task_struct', 'il_prev'),
4523                                    ('task_struct', 'mempolicy')])},
4524                    {'call': 'rt_sigtimedwait',
4525                     'reason': set([('task_struct', 'il_prev'),
4526                                    ('task_struct', 'mempolicy')])},
4527                    {'call': 'msgrcv',
4528                     'reason': set([('task_struct', 'il_prev'),
4529                                    ('task_struct', 'mempolicy')])},
4530                    {'call': 'kill',
4531                     'reason': set([('task_struct', 'il_prev'),
4532                                    ('task_struct', 'mempolicy')])},
4533                    {'call': 'sched_getaffinity',
4534                     'reason': set([('task_struct', 'il_prev'),
4535                                    ('task_struct', 'mempolicy')])},
4536                    {'call': 'sched_setparam',
4537                     'reason': set([('task_struct', 'il_prev'),
4538                                    ('task_struct', 'mempolicy')])},
4539                    {'call': 'ioprio_set',
4540                     'reason': set([('task_struct', 'il_prev'),
4541                                    ('task_struct', 'mempolicy')])},
4542                    {'call': 'remap_file_pages',
4543                     'reason': set([('vm_area_struct', 'vm_ops')])},
4544                    {'call': 'getppid',
4545                     'reason': set([('task_struct', 'il_prev'),
4546                                    ('task_struct', 'mempolicy')])},
4547                    {'call': 'mq_timedreceive',
```



```
4548                     'reason': set([('task_struct', 'il_prev'),
4549                                    ('task_struct', 'mempolicy')])},
4550                    {'call': 'capget',
4551                     'reason': set([('task_struct', 'il_prev'),
4552                                    ('task_struct', 'mempolicy')])},
4553                    {'call': 'sched_setaffinity',
4554                     'reason': set([('task_struct', 'il_prev'),
4555                                    ('task_struct', 'mempolicy')])},
4556                    {'call': 'signal',
4557                     'reason': set([('task_struct', 'il_prev'),
4558                                    ('task_struct', 'mempolicy')])},
4559                    {'call': 'semtimedop',
4560                     'reason': set([('task_struct', 'il_prev'),
4561                                    ('task_struct', 'mempolicy')])},
4562                    {'call': 'umount',
4563                     'reason': set([('task_struct', 'il_prev'),
4564                                    ('task_struct', 'mempolicy')])},
4565                    {'call': 'sched_rr_get_interval',
4566                     'reason': set([('task_struct', 'il_prev'),
4567                                    ('task_struct', 'mempolicy')])},
4568                    {'call': 'rt_sigprocmask',
4569                     'reason': set([('task_struct', 'il_prev'),
4570                                    ('task_struct', 'mempolicy')])},
4571                    {'call': 'setsid',
4572                     'reason': set([('task_struct', 'il_prev'),
4573                                    ('task_struct', 'mempolicy')])},
4574                    {'call': 'sigaltstack',
4575                     'reason': set([('task_struct', 'il_prev'),
4576                                    ('task_struct', 'mempolicy')])},
4577                    {'call': 'sched_setattr',
4578                     'reason': set([('task_struct', 'il_prev'),
4579                                    ('task_struct', 'mempolicy')])},
4580                    {'call': 'migrate_pages',
4581                     'reason': set([('task_struct', 'il_prev'),
4582                                    ('task_struct', 'mempolicy')])},
4583                    {'call': 'getitimer',
4584                     'reason': set([('task_struct', 'il_prev'),
4585                                    ('task_struct', 'mempolicy')])},
4586                    {'call': 'setpgid',
4587                     'reason': set([('task_struct', 'il_prev'),
4588                                    ('task_struct', 'mempolicy')])},
4589                    {'call': 'getsid',
4590                     'reason': set([('task_struct', 'il_prev'),
4591                                    ('task_struct', 'mempolicy')])},
4592                    {'call': 'prlimit64',
4593                     'reason': set([('task_struct', 'il_prev'),
4594                                    ('task_struct', 'mempolicy')])},
4595                    {'call': 'set_mempolicy',
4596                     'reason': set([('mempolicy', 'mode'),
4597                                    ('task_struct', 'il_prev'),
4598                                    ('task_struct', 'mempolicy')])},
4599                    {'call': 'perf_event_open',
4600                     'reason': set([('task_struct', 'il_prev'),
4601                                    ('task_struct', 'mempolicy')])},
4602                    {'call': 'shmdt',
4603                     'reason': set([('vm_area_struct', 'vm_ops')])},
4604                    {'call': 'rt_sigaction',
4605                     'reason': set([('task_struct', 'il_prev'),
4606                                    ('task_struct', 'mempolicy')])},
4607                    {'call': 'getpgid',
4608                     'reason': set([('task_struct', 'il_prev'),
4609                                    ('task_struct', 'mempolicy')])},
4610                    {'call': 'brk',
4611                     'reason': set([('vm_area_struct', 'vm_ops')])},
4612                    {'call': 'getpriority',
4613                     'reason': set([('task_struct', 'il_prev'),
```

```
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'sigaction',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'setns',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'fork',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'get_mempolicy',
                     'reason': set([('mempolicy', 'mode'),
                                    ('vm_area_struct', 'vm_ops')])},
                    {'call': 'get_robust_list',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'ptrace',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'munlockall',
                     'reason': set([('vm_area_struct', 'vm_ops')])},
                    {'call': 'pkey_mprotect',
                     'reason': set([('vm_area_struct', 'vm_ops')])},
                    {'call': 'madvise',
                     'reason': set([('vm_area_struct', 'vm_ops')])},
                    {'call': 'sched_getattr',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'getrusage',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'setitimer',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'ioprio_get',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'vfork',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'mprotect',
                     'reason': set([('vm_area_struct', 'vm_ops')])},
                    {'call': 'mremap',
                     'reason': set([('vm_area_struct', 'vm_ops')])},
                    {'call': 'mbind', 'reason': set([('mempolicy', 'mode')])},
                    {'call': 'prctl',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy'),
                                    ('vm_area_struct', 'vm_ops')])},
                    {'call': 'move_pages',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'munlock',
                     'reason': set([('vm_area_struct', 'vm_ops')])},
                    {'call': 'setpriority',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
```



```
                    {'call': 'mincore',
                     'reason': set([('vm_area_struct', 'vm_ops')])},
                    {'call': 'clone',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'sched_getparam',
                     'reason': set([('task_struct', 'il_prev'),
                                    ('task_struct', 'mempolicy')])},
                    {'call': 'mlockall',
                     'reason': set([('vm_area_struct', 'vm_ops')])}],
 'getcwd': [{'call': 'eventfd2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mq_unlink',
             'reason': set([('dentry', 'd_parent'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'swapoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'pivot_root',
             'reason': set([('dentry', 'd_parent'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'memfd_create',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'remap_file_pages',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'dup3',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'unshare',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'umount', 'reason': set([('vfsmount', 'mnt_root')])},
            {'call': 'mkdirat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'epoll_create1',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'epoll_ctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'flock',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'openat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'lookup_dcookie',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'uselib',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'renameat2', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'accept4',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'socketpair',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'getcwd',
             'reason': set([('dentry', 'd_parent'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'ftruncate', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'shmat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mknodat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'socket',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'symlinkat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'pipe2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'perf_event_open',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'linkat', 'reason': set([('dentry', 'd_parent')])},
```

```
4746             {'call': 'shmdt',
4747              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4748             {'call': 'quotactl',
4749              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4750             {'call': 'acct',
4751              'reason': set([('path', 'dentry'),
4752                             ('path', 'mnt'),
4753                             ('vfsmount', 'mnt_root')])},
4754             {'call': 'open',
4755              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4756             {'call': 'unlink', 'reason': set([('dentry', 'd_parent')])},
4757             {'call': 'rmdir', 'reason': set([('dentry', 'd_parent')])},
4758             {'call': 'dup',
4759              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4760             {'call': 'setns',
4761              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4762             {'call': 'shmctl',
4763              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4764             {'call': 'swapon',
4765              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4766             {'call': 'mmap_pgoff',
4767              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
4768             {'call': 'mq_open',
4769              'reason': set([('dentry', 'd_parent'),
4770                             ('path', 'dentry'),
4771                             ('path', 'mnt'),
4772                             ('vfsmount', 'mnt_root')])},
4773             {'call': 'unlinkat', 'reason': set([('dentry', 'd_parent')])},
4774             {'call': 'open_by_handle_at',
4775              'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
4776  'getdents': [{'call': 'syncfs',
4777                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4778               {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
4779               {'call': 'rt_sigtimedwait',
4780                'reason': set([('mm_segment_t', 'seg')])},
4781               {'call': 'vmsplice',
4782                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4783               {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
4784               {'call': 'pwritev64',
4785                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4786               {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
4787               {'call': 'fremovexattr',
4788                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4789               {'call': 'readahead',
4790                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4791               {'call': 'getdents',
4792                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4793               {'call': 'sched_getaffinity',
4794                'reason': set([('mm_segment_t', 'seg')])},
4795               {'call': 'writev',
4796                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4797               {'call': 'preadv64',
4798                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4799               {'call': 'sched_setparam',
4800                'reason': set([('mm_segment_t', 'seg')])},
4801               {'call': 'fchmod',
4802                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4803               {'call': 'pread64',
4804                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4805               {'call': 'signalfd4',
4806                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4807               {'call': 'ioprio_set',
4808                'reason': set([('mm_segment_t', 'seg')])},
4809               {'call': 'read',
4810                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4811               {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
```



```
4812               {'call': 'fchown',
4813                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4814               {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
4815               {'call': 'mq_timedreceive',
4816                'reason': set([('fd', 'file'),
4817                               ('fd', 'flags'),
4818                               ('mm_segment_t', 'seg')])},
4819               {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
4820               {'call': 'utime',
4821                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4822               {'call': 'sched_setaffinity',
4823                'reason': set([('mm_segment_t', 'seg')])},
4824               {'call': 'fsync',
4825                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4826               {'call': 'bpf',
4827                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4828               {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
4829               {'call': 'semtimedop',
4830                'reason': set([('mm_segment_t', 'seg')])},
4831               {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
4832               {'call': 'recvfrom',
4833                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4834               {'call': 'fsetxattr',
4835                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4836               {'call': 'sendto',
4837                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4838               {'call': 'sched_rr_get_interval',
4839                'reason': set([('mm_segment_t', 'seg')])},
4840               {'call': 'tee',
4841                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4842               {'call': 'sync_file_range',
4843                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4844               {'call': 'lseek',
4845                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4846               {'call': 'connect',
4847                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4848               {'call': 'getsockname',
4849                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4850               {'call': 'epoll_ctl',
4851                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4852               {'call': 'flock',
4853                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4854               {'call': 'pwritev',
4855                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4856               {'call': 'fchdir',
4857                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4858               {'call': 'rt_sigprocmask',
4859                'reason': set([('mm_segment_t', 'seg')])},
4860               {'call': 'accept4',
4861                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4862               {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
4863               {'call': 'sigaltstack',
4864                'reason': set([('mm_segment_t', 'seg')])},
4865               {'call': 'sched_setattr',
4866                'reason': set([('mm_segment_t', 'seg')])},
4867               {'call': 'old_readdir',
4868                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4869               {'call': 'inotify_rm_watch',
4870                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4871               {'call': 'utimensat',
4872                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4873               {'call': 'migrate_pages',
4874                'reason': set([('mm_segment_t', 'seg')])},
4875               {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
4876               {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
4877               {'call': 'inotify_add_watch',
```

```
4878                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4879               {'call': 'preadv2',
4880                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4881               {'call': 'splice',
4882                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4883               {'call': 'ftruncate',
4884                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4885               {'call': 'preadv',
4886                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4887               {'call': 'getpeername',
4888                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4889               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
4890               {'call': 'setsockopt',
4891                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4892               {'call': 'fcntl',
4893                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4894               {'call': 'ioctl',
4895                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4896               {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
4897               {'call': 'pwrite64',
4898                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4899               {'call': 'perf_event_open',
4900                'reason': set([('fd', 'file'),
4901                               ('fd', 'flags'),
4902                               ('mm_segment_t', 'seg')])},
4903               {'call': 'pwritev64v2',
4904                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4905               {'call': 'rt_sigaction',
4906                'reason': set([('mm_segment_t', 'seg')])},
4907               {'call': 'futimesat',
4908                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4909               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
4910               {'call': 'pwritev2',
4911                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4912               {'call': 'shutdown',
4913                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4914               {'call': 'getsockopt',
4915                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4916               {'call': 'getpriority',
4917                'reason': set([('mm_segment_t', 'seg')])},
4918               {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
4919               {'call': 'mq_getsetattr',
4920                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4921               {'call': 'fdatasync',
4922                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4923               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
4924               {'call': 'getdents64',
4925                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4926               {'call': 'listen',
4927                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4928               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
4929               {'call': 'get_robust_list',
4930                'reason': set([('mm_segment_t', 'seg')])},
4931               {'call': 'copy_file_range',
4932                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4933               {'call': 'mq_timedsend',
4934                'reason': set([('fd', 'file'),
4935                               ('fd', 'flags'),
4936                               ('mm_segment_t', 'seg')])},
4937               {'call': 'sched_getscheduler',
4938                'reason': set([('mm_segment_t', 'seg')])},
4939               {'call': 'fgetxattr',
4940                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4941               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
4942               {'call': 'fcntl64',
4943                'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
4944               {'call': 'fallocate',
4945                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4946               {'call': 'epoll_wait',
4947                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4948               {'call': 'sched_getattr',
4949                'reason': set([('mm_segment_t', 'seg')])},
4950               {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
4951               {'call': 'sched_setscheduler',
4952                'reason': set([('mm_segment_t', 'seg')])},
4953               {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
4954               {'call': 'ioprio_get',
4955                'reason': set([('mm_segment_t', 'seg')])},
4956               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
4957               {'call': 'llseek',
4958                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4959               {'call': 'preadv64v2',
4960                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4961               {'call': 'readv',
4962                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4963               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
4964               {'call': 'move_pages',
4965                'reason': set([('mm_segment_t', 'seg')])},
4966               {'call': 'fstatfs',
4967                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4968               {'call': 'fstatfs64',
4969                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4970               {'call': 'write',
4971                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4972               {'call': 'setpriority',
4973                'reason': set([('mm_segment_t', 'seg')])},
4974               {'call': 'mq_notify',
4975                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4976               {'call': 'sendfile',
4977                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4978               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
4979               {'call': 'sched_getparam',
4980                'reason': set([('mm_segment_t', 'seg')])},
4981               {'call': 'bind',
4982                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4983               {'call': 'flistxattr',
4984                'reason': set([('fd', 'file'), ('fd', 'flags')])},
4985               {'call': 'sendfile64',
4986                'reason': set([('fd', 'file'), ('fd', 'flags')])}],
4987  'getdents64': [{'call': 'syncfs',
4988                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
4989                 {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
4990                 {'call': 'rt_sigtimedwait',
4991                  'reason': set([('mm_segment_t', 'seg')])},
4992                 {'call': 'vmsplice',
4993                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
4994                 {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
4995                 {'call': 'pwritev64',
4996                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
4997                 {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
4998                 {'call': 'fremovexattr',
4999                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
5000                 {'call': 'readahead',
5001                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
5002                 {'call': 'getdents',
5003                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
5004                 {'call': 'sched_getaffinity',
5005                  'reason': set([('mm_segment_t', 'seg')])},
5006                 {'call': 'writev',
5007                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
5008                 {'call': 'preadv64',
5009                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
5010                {'call': 'sched_setparam',
5011                 'reason': set([('mm_segment_t', 'seg')])},
5012                {'call': 'fchmod',
5013                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5014                {'call': 'pread64',
5015                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5016                {'call': 'signalfd4',
5017                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5018                {'call': 'ioprio_set',
5019                 'reason': set([('mm_segment_t', 'seg')])},
5020                {'call': 'read',
5021                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5022                {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
5023                {'call': 'fchown',
5024                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5025                {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
5026                {'call': 'mq_timedreceive',
5027                 'reason': set([('fd', 'file'),
5028                                ('fd', 'flags'),
5029                                ('mm_segment_t', 'seg')])},
5030                {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
5031                {'call': 'utime',
5032                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5033                {'call': 'sched_setaffinity',
5034                 'reason': set([('mm_segment_t', 'seg')])},
5035                {'call': 'fsync',
5036                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5037                {'call': 'bpf',
5038                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5039                {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
5040                {'call': 'semtimedop',
5041                 'reason': set([('mm_segment_t', 'seg')])},
5042                {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
5043                {'call': 'recvfrom',
5044                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5045                {'call': 'fsetxattr',
5046                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5047                {'call': 'sendto',
5048                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5049                {'call': 'sched_rr_get_interval',
5050                 'reason': set([('mm_segment_t', 'seg')])},
5051                {'call': 'tee',
5052                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5053                {'call': 'sync_file_range',
5054                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5055                {'call': 'lseek',
5056                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5057                {'call': 'connect',
5058                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5059                {'call': 'getsockname',
5060                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5061                {'call': 'epoll_ctl',
5062                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5063                {'call': 'flock',
5064                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5065                {'call': 'pwritev',
5066                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5067                {'call': 'fchdir',
5068                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5069                {'call': 'rt_sigprocmask',
5070                 'reason': set([('mm_segment_t', 'seg')])},
5071                {'call': 'accept4',
5072                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5073                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
5074                {'call': 'sigaltstack',
5075                 'reason': set([('mm_segment_t', 'seg')])},
```



```
5076                {'call': 'sched_setattr',
5077                 'reason': set([('mm_segment_t', 'seg')])},
5078                {'call': 'old_readdir',
5079                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5080                {'call': 'inotify_rm_watch',
5081                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5082                {'call': 'utimensat',
5083                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5084                {'call': 'migrate_pages',
5085                 'reason': set([('mm_segment_t', 'seg')])},
5086                {'call': 'getitimer',
5087                 'reason': set([('mm_segment_t', 'seg')])},
5088                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
5089                {'call': 'inotify_add_watch',
5090                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5091                {'call': 'preadv2',
5092                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5093                {'call': 'splice',
5094                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5095                {'call': 'ftruncate',
5096                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5097                {'call': 'preadv',
5098                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5099                {'call': 'getpeername',
5100                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5101                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
5102                {'call': 'setsockopt',
5103                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5104                {'call': 'fcntl',
5105                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5106                {'call': 'ioctl',
5107                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5108                {'call': 'prlimit64',
5109                 'reason': set([('mm_segment_t', 'seg')])},
5110                {'call': 'pwrite64',
5111                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5112                {'call': 'perf_event_open',
5113                 'reason': set([('fd', 'file'),
5114                                ('fd', 'flags'),
5115                                ('mm_segment_t', 'seg')])},
5116                {'call': 'pwritev64v2',
5117                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5118                {'call': 'rt_sigaction',
5119                 'reason': set([('mm_segment_t', 'seg')])},
5120                {'call': 'futimesat',
5121                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5122                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
5123                {'call': 'pwritev2',
5124                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5125                {'call': 'shutdown',
5126                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5127                {'call': 'getsockopt',
5128                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5129                {'call': 'getpriority',
5130                 'reason': set([('mm_segment_t', 'seg')])},
5131                {'call': 'sigaction',
5132                 'reason': set([('mm_segment_t', 'seg')])},
5133                {'call': 'mq_getsetattr',
5134                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5135                {'call': 'fdatasync',
5136                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5137                {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
5138                {'call': 'getdents64',
5139                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
5140                {'call': 'listen',
5141                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'get_robust_list',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'copy_file_range',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mq_timedsend',
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('mm_segment_t', 'seg')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fgetxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fcntl64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fallocate',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'epoll_wait',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sched_getattr',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'getrusage',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'setitimer',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'ioprio_get',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'llseek',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'preadv64v2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'readv',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'move_pages',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fstatfs',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fstatfs64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'write',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'setpriority',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_notify',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sendfile',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'sched_getparam',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'bind',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'flistxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sendfile64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
 'getegid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
             {'call': 'rt_sigtimedwait',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
```



```
             {'call': 'sched_getaffinity',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_setparam',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
             {'call': 'mq_timedreceive',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_setaffinity',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_rr_get_interval',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'rt_sigprocmask',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
             {'call': 'sigaltstack',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_setattr',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'migrate_pages',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
             {'call': 'perf_event_open',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'rt_sigaction',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
             {'call': 'getpriority',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
             {'call': 'get_robust_list',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'mq_timedsend',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_getscheduler',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_getattr',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_setscheduler',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
             {'call': 'setpriority',
              'reason': set([('task_struct', 'cred')])},
             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
             {'call': 'sched_getparam',
              'reason': set([('task_struct', 'cred')])}],
 'getegid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
```

```
5274                {'call': 'sched_getaffinity',
5275                 'reason': set([('task_struct', 'cred')])},
5276                {'call': 'sched_setparam',
5277                 'reason': set([('task_struct', 'cred')])},
5278                {'call': 'ioprio_set',
5279                 'reason': set([('task_struct', 'cred')])},
5280                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
5281                {'call': 'mq_timedreceive',
5282                 'reason': set([('task_struct', 'cred')])},
5283                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
5284                {'call': 'sched_setaffinity',
5285                 'reason': set([('task_struct', 'cred')])},
5286                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
5287                {'call': 'semtimedop',
5288                 'reason': set([('task_struct', 'cred')])},
5289                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
5290                {'call': 'sched_rr_get_interval',
5291                 'reason': set([('task_struct', 'cred')])},
5292                {'call': 'rt_sigprocmask',
5293                 'reason': set([('task_struct', 'cred')])},
5294                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
5295                {'call': 'sigaltstack',
5296                 'reason': set([('task_struct', 'cred')])},
5297                {'call': 'sched_setattr',
5298                 'reason': set([('task_struct', 'cred')])},
5299                {'call': 'migrate_pages',
5300                 'reason': set([('task_struct', 'cred')])},
5301                {'call': 'getitimer',
5302                 'reason': set([('task_struct', 'cred')])},
5303                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
5304                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
5305                {'call': 'prlimit64',
5306                 'reason': set([('task_struct', 'cred')])},
5307                {'call': 'perf_event_open',
5308                 'reason': set([('task_struct', 'cred')])},
5309                {'call': 'rt_sigaction',
5310                 'reason': set([('task_struct', 'cred')])},
5311                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
5312                {'call': 'getpriority',
5313                 'reason': set([('task_struct', 'cred')])},
5314                {'call': 'sigaction',
5315                 'reason': set([('task_struct', 'cred')])},
5316                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
5317                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
5318                {'call': 'get_robust_list',
5319                 'reason': set([('task_struct', 'cred')])},
5320                {'call': 'mq_timedsend',
5321                 'reason': set([('task_struct', 'cred')])},
5322                {'call': 'sched_getscheduler',
5323                 'reason': set([('task_struct', 'cred')])},
5324                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
5325                {'call': 'sched_getattr',
5326                 'reason': set([('task_struct', 'cred')])},
5327                {'call': 'getrusage',
5328                 'reason': set([('task_struct', 'cred')])},
5329                {'call': 'sched_setscheduler',
5330                 'reason': set([('task_struct', 'cred')])},
5331                {'call': 'setitimer',
5332                 'reason': set([('task_struct', 'cred')])},
5333                {'call': 'ioprio_get',
5334                 'reason': set([('task_struct', 'cred')])},
5335                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
5336                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
5337                {'call': 'move_pages',
5338                 'reason': set([('task_struct', 'cred')])},
5339                {'call': 'setpriority',
```



```
5340                 'reason': set([('task_struct', 'cred')])},
5341                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
5342                {'call': 'sched_getparam',
5343                 'reason': set([('task_struct', 'cred')])}],
5344  'geteuid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
5345              {'call': 'rt_sigtimedwait',
5346               'reason': set([('task_struct', 'cred')])},
5347              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
5348              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
5349              {'call': 'sched_getaffinity',
5350               'reason': set([('task_struct', 'cred')])},
5351              {'call': 'sched_setparam',
5352               'reason': set([('task_struct', 'cred')])},
5353              {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
5354              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
5355              {'call': 'mq_timedreceive',
5356               'reason': set([('task_struct', 'cred')])},
5357              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
5358              {'call': 'sched_setaffinity',
5359               'reason': set([('task_struct', 'cred')])},
5360              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
5361              {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
5362              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
5363              {'call': 'sched_rr_get_interval',
5364               'reason': set([('task_struct', 'cred')])},
5365              {'call': 'rt_sigprocmask',
5366               'reason': set([('task_struct', 'cred')])},
5367              {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
5368              {'call': 'sigaltstack',
5369               'reason': set([('task_struct', 'cred')])},
5370              {'call': 'sched_setattr',
5371               'reason': set([('task_struct', 'cred')])},
5372              {'call': 'migrate_pages',
5373               'reason': set([('task_struct', 'cred')])},
5374              {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
5375              {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
5376              {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
5377              {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
5378              {'call': 'perf_event_open',
5379               'reason': set([('task_struct', 'cred')])},
5380              {'call': 'rt_sigaction',
5381               'reason': set([('task_struct', 'cred')])},
5382              {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
5383              {'call': 'getpriority',
5384               'reason': set([('task_struct', 'cred')])},
5385              {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
5386              {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
5387              {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
5388              {'call': 'get_robust_list',
5389               'reason': set([('task_struct', 'cred')])},
5390              {'call': 'mq_timedsend',
5391               'reason': set([('task_struct', 'cred')])},
5392              {'call': 'sched_getscheduler',
5393               'reason': set([('task_struct', 'cred')])},
5394              {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
5395              {'call': 'sched_getattr',
5396               'reason': set([('task_struct', 'cred')])},
5397              {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
5398              {'call': 'sched_setscheduler',
5399               'reason': set([('task_struct', 'cred')])},
5400              {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
5401              {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
5402              {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
5403              {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
5404              {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
5405              {'call': 'setpriority',
```

```
5406                'reason': set([('task_struct', 'cred')])},
5407               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
5408               {'call': 'sched_getparam',
5409                'reason': set([('task_struct', 'cred')])}],
5410  'geteuid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
5411                {'call': 'rt_sigtimedwait',
5412                 'reason': set([('task_struct', 'cred')])},
5413                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
5414                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
5415                {'call': 'sched_getaffinity',
5416                 'reason': set([('task_struct', 'cred')])},
5417                {'call': 'sched_setparam',
5418                 'reason': set([('task_struct', 'cred')])},
5419                {'call': 'ioprio_set',
5420                 'reason': set([('task_struct', 'cred')])},
5421                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
5422                {'call': 'mq_timedreceive',
5423                 'reason': set([('task_struct', 'cred')])},
5424                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
5425                {'call': 'sched_setaffinity',
5426                 'reason': set([('task_struct', 'cred')])},
5427                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
5428                {'call': 'semtimedop',
5429                 'reason': set([('task_struct', 'cred')])},
5430                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
5431                {'call': 'sched_rr_get_interval',
5432                 'reason': set([('task_struct', 'cred')])},
5433                {'call': 'rt_sigprocmask',
5434                 'reason': set([('task_struct', 'cred')])},
5435                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
5436                {'call': 'sigaltstack',
5437                 'reason': set([('task_struct', 'cred')])},
5438                {'call': 'sched_setattr',
5439                 'reason': set([('task_struct', 'cred')])},
5440                {'call': 'migrate_pages',
5441                 'reason': set([('task_struct', 'cred')])},
5442                {'call': 'getitimer',
5443                 'reason': set([('task_struct', 'cred')])},
5444                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
5445                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
5446                {'call': 'prlimit64',
5447                 'reason': set([('task_struct', 'cred')])},
5448                {'call': 'perf_event_open',
5449                 'reason': set([('task_struct', 'cred')])},
5450                {'call': 'rt_sigaction',
5451                 'reason': set([('task_struct', 'cred')])},
5452                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
5453                {'call': 'getpriority',
5454                 'reason': set([('task_struct', 'cred')])},
5455                {'call': 'sigaction',
5456                 'reason': set([('task_struct', 'cred')])},
5457                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
5458                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
5459                {'call': 'get_robust_list',
5460                 'reason': set([('task_struct', 'cred')])},
5461                {'call': 'mq_timedsend',
5462                 'reason': set([('task_struct', 'cred')])},
5463                {'call': 'sched_getscheduler',
5464                 'reason': set([('task_struct', 'cred')])},
5465                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
5466                {'call': 'sched_getattr',
5467                 'reason': set([('task_struct', 'cred')])},
5468                {'call': 'getrusage',
5469                 'reason': set([('task_struct', 'cred')])},
5470                {'call': 'sched_setscheduler',
5471                 'reason': set([('task_struct', 'cred')])},
```



```
5472                {'call': 'setitimer',
5473                 'reason': set([('task_struct', 'cred')])},
5474                {'call': 'ioprio_get',
5475                 'reason': set([('task_struct', 'cred')])},
5476                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
5477                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
5478                {'call': 'move_pages',
5479                 'reason': set([('task_struct', 'cred')])},
5480                {'call': 'setpriority',
5481                 'reason': set([('task_struct', 'cred')])},
5482                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
5483                {'call': 'sched_getparam',
5484                 'reason': set([('task_struct', 'cred')])}],
5485  'getgid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
5486             {'call': 'rt_sigtimedwait',
5487              'reason': set([('task_struct', 'cred')])},
5488             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
5489             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
5490             {'call': 'sched_getaffinity',
5491              'reason': set([('task_struct', 'cred')])},
5492             {'call': 'sched_setparam',
5493              'reason': set([('task_struct', 'cred')])},
5494             {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
5495             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
5496             {'call': 'mq_timedreceive',
5497              'reason': set([('task_struct', 'cred')])},
5498             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
5499             {'call': 'sched_setaffinity',
5500              'reason': set([('task_struct', 'cred')])},
5501             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
5502             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
5503             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
5504             {'call': 'sched_rr_get_interval',
5505              'reason': set([('task_struct', 'cred')])},
5506             {'call': 'rt_sigprocmask',
5507              'reason': set([('task_struct', 'cred')])},
5508             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
5509             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
5510             {'call': 'sched_setattr',
5511              'reason': set([('task_struct', 'cred')])},
5512             {'call': 'migrate_pages',
5513              'reason': set([('task_struct', 'cred')])},
5514             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
5515             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
5516             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
5517             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
5518             {'call': 'perf_event_open',
5519              'reason': set([('task_struct', 'cred')])},
5520             {'call': 'rt_sigaction',
5521              'reason': set([('task_struct', 'cred')])},
5522             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
5523             {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
5524             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
5525             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
5526             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
5527             {'call': 'get_robust_list',
5528              'reason': set([('task_struct', 'cred')])},
5529             {'call': 'mq_timedsend',
5530              'reason': set([('task_struct', 'cred')])},
5531             {'call': 'sched_getscheduler',
5532              'reason': set([('task_struct', 'cred')])},
5533             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
5534             {'call': 'sched_getattr',
5535              'reason': set([('task_struct', 'cred')])},
5536             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
5537             {'call': 'sched_setscheduler',
```

```
5538              'reason': set([('task_struct', 'cred')])},
5539             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
5540             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
5541             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
5542             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
5543             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
5544             {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
5545             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
5546             {'call': 'sched_getparam',
5547              'reason': set([('task_struct', 'cred')])}],
5548  'getgid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
5549               {'call': 'rt_sigtimedwait',
5550                'reason': set([('task_struct', 'cred')])},
5551               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
5552               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
5553               {'call': 'sched_getaffinity',
5554                'reason': set([('task_struct', 'cred')])},
5555               {'call': 'sched_setparam',
5556                'reason': set([('task_struct', 'cred')])},
5557               {'call': 'ioprio_set',
5558                'reason': set([('task_struct', 'cred')])},
5559               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
5560               {'call': 'mq_timedreceive',
5561                'reason': set([('task_struct', 'cred')])},
5562               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
5563               {'call': 'sched_setaffinity',
5564                'reason': set([('task_struct', 'cred')])},
5565               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
5566               {'call': 'semtimedop',
5567                'reason': set([('task_struct', 'cred')])},
5568               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
5569               {'call': 'sched_rr_get_interval',
5570                'reason': set([('task_struct', 'cred')])},
5571               {'call': 'rt_sigprocmask',
5572                'reason': set([('task_struct', 'cred')])},
5573               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
5574               {'call': 'sigaltstack',
5575                'reason': set([('task_struct', 'cred')])},
5576               {'call': 'sched_setattr',
5577                'reason': set([('task_struct', 'cred')])},
5578               {'call': 'migrate_pages',
5579                'reason': set([('task_struct', 'cred')])},
5580               {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
5581               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
5582               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
5583               {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
5584               {'call': 'perf_event_open',
5585                'reason': set([('task_struct', 'cred')])},
5586               {'call': 'rt_sigaction',
5587                'reason': set([('task_struct', 'cred')])},
5588               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
5589               {'call': 'getpriority',
5590                'reason': set([('task_struct', 'cred')])},
5591               {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
5592               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
5593               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
5594               {'call': 'get_robust_list',
5595                'reason': set([('task_struct', 'cred')])},
5596               {'call': 'mq_timedsend',
5597                'reason': set([('task_struct', 'cred')])},
5598               {'call': 'sched_getscheduler',
5599                'reason': set([('task_struct', 'cred')])},
5600               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
5601               {'call': 'sched_getattr',
5602                'reason': set([('task_struct', 'cred')])},
5603               {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
```



```
5604              {'call': 'sched_setscheduler',
5605               'reason': set([('task_struct', 'cred')])},
5606              {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
5607              {'call': 'ioprio_get',
5608               'reason': set([('task_struct', 'cred')])},
5609              {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
5610              {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
5611              {'call': 'move_pages',
5612               'reason': set([('task_struct', 'cred')])},
5613              {'call': 'setpriority',
5614               'reason': set([('task_struct', 'cred')])},
5615              {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
5616              {'call': 'sched_getparam',
5617               'reason': set([('task_struct', 'cred')])}],
5618  'getgroups': [{'call': 'keyctl',
5619                 'reason': set([('cred', 'group_info'),
5620                                ('task_struct', 'cred')])},
5621                {'call': 'rt_sigtimedwait',
5622                 'reason': set([('task_struct', 'cred')])},
5623                {'call': 'setfsuid', 'reason': set([('cred', 'group_info')])},
5624                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
5625                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
5626                {'call': 'getresuid16',
5627                 'reason': set([('cred', 'group_info')])},
5628                {'call': 'getresgid', 'reason': set([('cred', 'group_info')])},
5629                {'call': 'sched_getaffinity',
5630                 'reason': set([('task_struct', 'cred')])},
5631                {'call': 'sched_setparam',
5632                 'reason': set([('task_struct', 'cred')])},
5633                {'call': 'setgid', 'reason': set([('cred', 'group_info')])},
5634                {'call': 'ioprio_set',
5635                 'reason': set([('cred', 'group_info'),
5636                                ('task_struct', 'cred')])},
5637                {'call': 'capset', 'reason': set([('cred', 'group_info')])},
5638                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
5639                {'call': 'mq_timedreceive',
5640                 'reason': set([('task_struct', 'cred')])},
5641                {'call': 'getresgid16',
5642                 'reason': set([('cred', 'group_info')])},
5643                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
5644                {'call': 'sched_setaffinity',
5645                 'reason': set([('cred', 'group_info'),
5646                                ('task_struct', 'cred')])},
5647                {'call': 'setfsgid', 'reason': set([('cred', 'group_info')])},
5648                {'call': 'unshare', 'reason': set([('cred', 'group_info')])},
5649                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
5650                {'call': 'setreuid', 'reason': set([('cred', 'group_info')])},
5651                {'call': 'semtimedop',
5652                 'reason': set([('task_struct', 'cred')])},
5653                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
5654                {'call': 'sched_rr_get_interval',
5655                 'reason': set([('task_struct', 'cred')])},
5656                {'call': 'epoll_create1',
5657                 'reason': set([('cred', 'group_info')])},
5658                {'call': 'getresuid', 'reason': set([('cred', 'group_info')])},
5659                {'call': 'rt_sigprocmask',
5660                 'reason': set([('task_struct', 'cred')])},
5661                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
5662                {'call': 'sigaltstack',
5663                 'reason': set([('task_struct', 'cred')])},
5664                {'call': 'sched_setattr',
5665                 'reason': set([('task_struct', 'cred')])},
5666                {'call': 'migrate_pages',
5667                 'reason': set([('cred', 'group_info'),
5668                                ('task_struct', 'cred')])},
5669                {'call': 'getitimer',
```

```
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setresgid', 'reason': set([('cred', 'group_info')])},
                {'call': 'setregid', 'reason': set([('cred', 'group_info')])},
                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'prlimit64',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getgroups16',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpriority',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'faccessat', 'reason': set([('cred', 'group_info')])},
                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setresuid', 'reason': set([('cred', 'group_info')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setuid', 'reason': set([('cred', 'group_info')])},
                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getgroups', 'reason': set([('cred', 'group_info')])},
                {'call': 'setpriority',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred')])}],
 'getgroups16': [{'call': 'keyctl',
                  'reason': set([('cred', 'group_info'),
                                 ('task_struct', 'cred')])},
                 {'call': 'rt_sigtimedwait',
                  'reason': set([('task_struct', 'cred')])},
                 {'call': 'setfsuid',
                  'reason': set([('cred', 'group_info')])},
                 {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
                 {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
                 {'call': 'getresuid16',
                  'reason': set([('cred', 'group_info')])},
                 {'call': 'getresgid',
```



```
                 'reason': set([('cred', 'group_info')])},
                {'call': 'sched_getaffinity',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setparam',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setgid', 'reason': set([('cred', 'group_info')])},
                {'call': 'ioprio_set',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'capset', 'reason': set([('cred', 'group_info')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getresgid16',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'setfsgid',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'unshare', 'reason': set([('cred', 'group_info')])},
                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setreuid',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'epoll_create1',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'getresuid',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'migrate_pages',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setresgid',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'setregid',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'prlimit64',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getgroups16',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpriority',
```

```
                  'reason': set([('cred', 'group_info'),
                                 ('task_struct', 'cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'faccessat',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'setgroups16',
                 'reason': set([('group_info', 'ngroups')])},
                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setresuid',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setuid', 'reason': set([('cred', 'group_info')])},
                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getgroups',
                 'reason': set([('cred', 'group_info')])},
                {'call': 'setpriority',
                 'reason': set([('cred', 'group_info'),
                                ('task_struct', 'cred')])},
                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setgroups',
                 'reason': set([('group_info', 'ngroups')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred')])}],
 'getitimer': [{'call': 'timer_create',
                'reason': set([('signal_struct', 'it_real_incr')])},
               {'call': 'exit_group',
                'reason': set([('signal_struct', 'it_real_incr')])},
               {'call': 'setitimer',
                'reason': set([('signal_struct', 'it_real_incr')])}],
 'getpeername': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
                 {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
                 {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
                 {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
                 {'call': 'readahead', 'reason': set([('fd', 'file')])},
                 {'call': 'getdents', 'reason': set([('fd', 'file')])},
                 {'call': 'writev', 'reason': set([('fd', 'file')])},
                 {'call': 'preadv64', 'reason': set([('fd', 'file')])},
                 {'call': 'fchmod', 'reason': set([('fd', 'file')])},
                 {'call': 'pread64', 'reason': set([('fd', 'file')])},
                 {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
                 {'call': 'read', 'reason': set([('fd', 'file')])},
                 {'call': 'fchown', 'reason': set([('fd', 'file')])},
                 {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
                 {'call': 'utime', 'reason': set([('fd', 'file')])},
```



```
                 {'call': 'fsync', 'reason': set([('fd', 'file')])},
                 {'call': 'bpf', 'reason': set([('fd', 'file')])},
                 {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
                 {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
                 {'call': 'sendto', 'reason': set([('fd', 'file')])},
                 {'call': 'tee', 'reason': set([('fd', 'file')])},
                 {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
                 {'call': 'lseek', 'reason': set([('fd', 'file')])},
                 {'call': 'connect', 'reason': set([('fd', 'file')])},
                 {'call': 'getsockname', 'reason': set([('fd', 'file')])},
                 {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
                 {'call': 'flock', 'reason': set([('fd', 'file')])},
                 {'call': 'pwritev', 'reason': set([('fd', 'file')])},
                 {'call': 'fchdir', 'reason': set([('fd', 'file')])},
                 {'call': 'accept4', 'reason': set([('fd', 'file')])},
                 {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
                 {'call': 'inotify_rm_watch',
                  'reason': set([('fd', 'file')])},
                 {'call': 'utimensat', 'reason': set([('fd', 'file')])},
                 {'call': 'inotify_add_watch',
                  'reason': set([('fd', 'file')])},
                 {'call': 'preadv2', 'reason': set([('fd', 'file')])},
                 {'call': 'splice', 'reason': set([('fd', 'file')])},
                 {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
                 {'call': 'preadv', 'reason': set([('fd', 'file')])},
                 {'call': 'getpeername', 'reason': set([('fd', 'file')])},
                 {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
                 {'call': 'fcntl', 'reason': set([('fd', 'file')])},
                 {'call': 'ioctl', 'reason': set([('fd', 'file')])},
                 {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
                 {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
                 {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
                 {'call': 'futimesat', 'reason': set([('fd', 'file')])},
                 {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
                 {'call': 'shutdown', 'reason': set([('fd', 'file')])},
                 {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
                 {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
                 {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
                 {'call': 'getdents64', 'reason': set([('fd', 'file')])},
                 {'call': 'listen', 'reason': set([('fd', 'file')])},
                 {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
                 {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
                 {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
                 {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
                 {'call': 'fallocate', 'reason': set([('fd', 'file')])},
                 {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
                 {'call': 'llseek', 'reason': set([('fd', 'file')])},
                 {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
                 {'call': 'readv', 'reason': set([('fd', 'file')])},
                 {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
                 {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
                 {'call': 'write', 'reason': set([('fd', 'file')])},
                 {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
                 {'call': 'sendfile', 'reason': set([('fd', 'file')])},
                 {'call': 'bind', 'reason': set([('fd', 'file')])},
                 {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
                 {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
 'getppid': [{'call': 'keyctl',
              'reason': set([('task_struct', 'real_parent')])},
             {'call': 'rt_sigtimedwait',
              'reason': set([('task_struct', 'real_parent')])},
             {'call': 'msgrcv',
              'reason': set([('task_struct', 'real_parent')])},
             {'call': 'kill',
              'reason': set([('task_struct', 'real_parent')])},
             {'call': 'sched_getaffinity',
```

```
5934                'reason': set([('task_struct', 'real_parent')])},
5935               {'call': 'sched_setparam',
5936                'reason': set([('task_struct', 'real_parent')])},
5937               {'call': 'ioprio_set',
5938                'reason': set([('task_struct', 'real_parent')])},
5939               {'call': 'getppid',
5940                'reason': set([('task_struct', 'real_parent')])},
5941               {'call': 'mq_timedreceive',
5942                'reason': set([('task_struct', 'real_parent')])},
5943               {'call': 'capget',
5944                'reason': set([('task_struct', 'real_parent')])},
5945               {'call': 'sched_setaffinity',
5946                'reason': set([('task_struct', 'real_parent')])},
5947               {'call': 'signal',
5948                'reason': set([('task_struct', 'real_parent')])},
5949               {'call': 'semtimedop',
5950                'reason': set([('task_struct', 'real_parent')])},
5951               {'call': 'umount',
5952                'reason': set([('task_struct', 'real_parent')])},
5953               {'call': 'sched_rr_get_interval',
5954                'reason': set([('task_struct', 'real_parent')])},
5955               {'call': 'rt_sigprocmask',
5956                'reason': set([('task_struct', 'real_parent')])},
5957               {'call': 'setsid',
5958                'reason': set([('task_struct', 'real_parent')])},
5959               {'call': 'sigaltstack',
5960                'reason': set([('task_struct', 'real_parent')])},
5961               {'call': 'sched_setattr',
5962                'reason': set([('task_struct', 'real_parent')])},
5963               {'call': 'migrate_pages',
5964                'reason': set([('task_struct', 'real_parent')])},
5965               {'call': 'getitimer',
5966                'reason': set([('task_struct', 'real_parent')])},
5967               {'call': 'setpgid',
5968                'reason': set([('task_struct', 'real_parent')])},
5969               {'call': 'getsid',
5970                'reason': set([('task_struct', 'real_parent')])},
5971               {'call': 'prlimit64',
5972                'reason': set([('task_struct', 'real_parent')])},
5973               {'call': 'perf_event_open',
5974                'reason': set([('task_struct', 'real_parent')])},
5975               {'call': 'rt_sigaction',
5976                'reason': set([('task_struct', 'real_parent')])},
5977               {'call': 'getpgid',
5978                'reason': set([('task_struct', 'real_parent')])},
5979               {'call': 'getpriority',
5980                'reason': set([('task_struct', 'real_parent')])},
5981               {'call': 'sigaction',
5982                'reason': set([('task_struct', 'real_parent')])},
5983               {'call': 'setns',
5984                'reason': set([('task_struct', 'real_parent')])},
5985               {'call': 'fork',
5986                'reason': set([('task_struct', 'real_parent')])},
5987               {'call': 'get_robust_list',
5988                'reason': set([('task_struct', 'real_parent')])},
5989               {'call': 'mq_timedsend',
5990                'reason': set([('task_struct', 'real_parent')])},
5991               {'call': 'sched_getscheduler',
5992                'reason': set([('task_struct', 'real_parent')])},
5993               {'call': 'ptrace',
5994                'reason': set([('task_struct', 'real_parent')])},
5995               {'call': 'sched_getattr',
5996                'reason': set([('task_struct', 'real_parent')])},
5997               {'call': 'getrusage',
5998                'reason': set([('task_struct', 'real_parent')])},
5999               {'call': 'sched_setscheduler',
```



```
6000                'reason': set([('task_struct', 'real_parent')])},
6001               {'call': 'setitimer',
6002                'reason': set([('task_struct', 'real_parent')])},
6003               {'call': 'ioprio_get',
6004                'reason': set([('task_struct', 'real_parent')])},
6005               {'call': 'vfork',
6006                'reason': set([('task_struct', 'real_parent')])},
6007               {'call': 'prctl',
6008                'reason': set([('task_struct', 'real_parent')])},
6009               {'call': 'move_pages',
6010                'reason': set([('task_struct', 'real_parent')])},
6011               {'call': 'setpriority',
6012                'reason': set([('task_struct', 'real_parent')])},
6013               {'call': 'clone',
6014                'reason': set([('task_struct', 'real_parent')])},
6015               {'call': 'sched_getparam',
6016                'reason': set([('task_struct', 'real_parent')])}],
6017  'getpriority': [{'call': 'keyctl',
6018                   'reason': set([('task_struct', 'cred'),
6019                                  ('task_struct', 'real_cred')])},
6020                  {'call': 'rt_sigtimedwait',
6021                   'reason': set([('task_struct', 'cred'),
6022                                  ('task_struct', 'real_cred')])},
6023                  {'call': 'msgrcv',
6024                   'reason': set([('task_struct', 'cred'),
6025                                  ('task_struct', 'real_cred')])},
6026                  {'call': 'kill',
6027                   'reason': set([('task_struct', 'cred'),
6028                                  ('task_struct', 'real_cred')])},
6029                  {'call': 'sched_getaffinity',
6030                   'reason': set([('task_struct', 'cred'),
6031                                  ('task_struct', 'real_cred')])},
6032                  {'call': 'sched_setparam',
6033                   'reason': set([('task_struct', 'cred'),
6034                                  ('task_struct', 'real_cred')])},
6035                  {'call': 'ioprio_set',
6036                   'reason': set([('task_struct', 'cred'),
6037                                  ('task_struct', 'real_cred')])},
6038                  {'call': 'getppid',
6039                   'reason': set([('task_struct', 'cred'),
6040                                  ('task_struct', 'real_cred')])},
6041                  {'call': 'mq_timedreceive',
6042                   'reason': set([('task_struct', 'cred'),
6043                                  ('task_struct', 'real_cred')])},
6044                  {'call': 'capget',
6045                   'reason': set([('task_struct', 'cred'),
6046                                  ('task_struct', 'real_cred')])},
6047                  {'call': 'sched_setaffinity',
6048                   'reason': set([('task_struct', 'cred'),
6049                                  ('task_struct', 'real_cred')])},
6050                  {'call': 'signal',
6051                   'reason': set([('task_struct', 'cred'),
6052                                  ('task_struct', 'real_cred')])},
6053                  {'call': 'setreuid', 'reason': set([('cred', 'uid')])},
6054                  {'call': 'semtimedop',
6055                   'reason': set([('task_struct', 'cred'),
6056                                  ('task_struct', 'real_cred')])},
6057                  {'call': 'umount',
6058                   'reason': set([('task_struct', 'cred'),
6059                                  ('task_struct', 'real_cred')])},
6060                  {'call': 'sched_rr_get_interval',
6061                   'reason': set([('task_struct', 'cred'),
6062                                  ('task_struct', 'real_cred')])},
6063                  {'call': 'rt_sigprocmask',
6064                   'reason': set([('task_struct', 'cred'),
6065                                  ('task_struct', 'real_cred')])},
```

```
6066                 {'call': 'setsid',
6067                  'reason': set([('task_struct', 'cred'),
6068                                 ('task_struct', 'real_cred')])},
6069                 {'call': 'sigaltstack',
6070                  'reason': set([('task_struct', 'cred'),
6071                                 ('task_struct', 'real_cred')])},
6072                 {'call': 'sched_setattr',
6073                  'reason': set([('task_struct', 'cred'),
6074                                 ('task_struct', 'real_cred')])},
6075                 {'call': 'migrate_pages',
6076                  'reason': set([('task_struct', 'cred'),
6077                                 ('task_struct', 'real_cred')])},
6078                 {'call': 'getitimer',
6079                  'reason': set([('task_struct', 'cred'),
6080                                 ('task_struct', 'real_cred')])},
6081                 {'call': 'setpgid',
6082                  'reason': set([('task_struct', 'cred'),
6083                                 ('task_struct', 'real_cred')])},
6084                 {'call': 'getsid',
6085                  'reason': set([('task_struct', 'cred'),
6086                                 ('task_struct', 'real_cred')])},
6087                 {'call': 'prlimit64',
6088                  'reason': set([('task_struct', 'cred'),
6089                                 ('task_struct', 'real_cred')])},
6090                 {'call': 'perf_event_open',
6091                  'reason': set([('task_struct', 'cred'),
6092                                 ('task_struct', 'real_cred')])},
6093                 {'call': 'rt_sigaction',
6094                  'reason': set([('task_struct', 'cred'),
6095                                 ('task_struct', 'real_cred')])},
6096                 {'call': 'getpgid',
6097                  'reason': set([('task_struct', 'cred'),
6098                                 ('task_struct', 'real_cred')])},
6099                 {'call': 'getpriority',
6100                  'reason': set([('task_struct', 'cred'),
6101                                 ('task_struct', 'real_cred')])},
6102                 {'call': 'sigaction',
6103                  'reason': set([('task_struct', 'cred'),
6104                                 ('task_struct', 'real_cred')])},
6105                 {'call': 'setns',
6106                  'reason': set([('task_struct', 'cred'),
6107                                 ('task_struct', 'real_cred')])},
6108                 {'call': 'fork',
6109                  'reason': set([('task_struct', 'cred'),
6110                                 ('task_struct', 'real_cred')])},
6111                 {'call': 'get_robust_list',
6112                  'reason': set([('task_struct', 'cred'),
6113                                 ('task_struct', 'real_cred')])},
6114                 {'call': 'mq_timedsend',
6115                  'reason': set([('task_struct', 'cred'),
6116                                 ('task_struct', 'real_cred')])},
6117                 {'call': 'sched_getscheduler',
6118                  'reason': set([('task_struct', 'cred'),
6119                                 ('task_struct', 'real_cred')])},
6120                 {'call': 'ptrace',
6121                  'reason': set([('task_struct', 'cred'),
6122                                 ('task_struct', 'real_cred')])},
6123                 {'call': 'sched_getattr',
6124                  'reason': set([('task_struct', 'cred'),
6125                                 ('task_struct', 'real_cred')])},
6126                 {'call': 'getrusage',
6127                  'reason': set([('task_struct', 'cred'),
6128                                 ('task_struct', 'real_cred')])},
6129                 {'call': 'sched_setscheduler',
6130                  'reason': set([('task_struct', 'cred'),
6131                                 ('task_struct', 'real_cred')])},
```



```
6132                 {'call': 'setresuid', 'reason': set([('cred', 'uid')])},
6133                 {'call': 'setitimer',
6134                  'reason': set([('task_struct', 'cred'),
6135                                 ('task_struct', 'real_cred')])},
6136                 {'call': 'ioprio_get',
6137                  'reason': set([('task_struct', 'cred'),
6138                                 ('task_struct', 'real_cred')])},
6139                 {'call': 'vfork',
6140                  'reason': set([('task_struct', 'cred'),
6141                                 ('task_struct', 'real_cred')])},
6142                 {'call': 'setuid', 'reason': set([('cred', 'uid')])},
6143                 {'call': 'prctl',
6144                  'reason': set([('task_struct', 'cred'),
6145                                 ('task_struct', 'real_cred')])},
6146                 {'call': 'move_pages',
6147                  'reason': set([('task_struct', 'cred'),
6148                                 ('task_struct', 'real_cred')])},
6149                 {'call': 'setpriority',
6150                  'reason': set([('task_struct', 'cred'),
6151                                 ('task_struct', 'real_cred')])},
6152                 {'call': 'clone',
6153                  'reason': set([('task_struct', 'cred'),
6154                                 ('task_struct', 'real_cred')])},
6155                 {'call': 'sched_getparam',
6156                  'reason': set([('task_struct', 'cred'),
6157                                 ('task_struct', 'real_cred')])}],
6158  'getresgid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
6159                {'call': 'rt_sigtimedwait',
6160                 'reason': set([('task_struct', 'cred')])},
6161                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
6162                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
6163                {'call': 'sched_getaffinity',
6164                 'reason': set([('task_struct', 'cred')])},
6165                {'call': 'sched_setparam',
6166                 'reason': set([('task_struct', 'cred')])},
6167                {'call': 'ioprio_set',
6168                 'reason': set([('task_struct', 'cred')])},
6169                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
6170                {'call': 'mq_timedreceive',
6171                 'reason': set([('task_struct', 'cred')])},
6172                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
6173                {'call': 'sched_setaffinity',
6174                 'reason': set([('task_struct', 'cred')])},
6175                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
6176                {'call': 'semtimedop',
6177                 'reason': set([('task_struct', 'cred')])},
6178                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
6179                {'call': 'sched_rr_get_interval',
6180                 'reason': set([('task_struct', 'cred')])},
6181                {'call': 'rt_sigprocmask',
6182                 'reason': set([('task_struct', 'cred')])},
6183                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
6184                {'call': 'sigaltstack',
6185                 'reason': set([('task_struct', 'cred')])},
6186                {'call': 'sched_setattr',
6187                 'reason': set([('task_struct', 'cred')])},
6188                {'call': 'migrate_pages',
6189                 'reason': set([('task_struct', 'cred')])},
6190                {'call': 'getitimer',
6191                 'reason': set([('task_struct', 'cred')])},
6192                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
6193                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
6194                {'call': 'prlimit64',
6195                 'reason': set([('task_struct', 'cred')])},
6196                {'call': 'perf_event_open',
6197                 'reason': set([('task_struct', 'cred')])},
```

```
6198                {'call': 'rt_sigaction',
6199                 'reason': set([('task_struct', 'cred')])},
6200                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
6201                {'call': 'getpriority',
6202                 'reason': set([('task_struct', 'cred')])},
6203                {'call': 'sigaction',
6204                 'reason': set([('task_struct', 'cred')])},
6205                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
6206                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
6207                {'call': 'get_robust_list',
6208                 'reason': set([('task_struct', 'cred')])},
6209                {'call': 'mq_timedsend',
6210                 'reason': set([('task_struct', 'cred')])},
6211                {'call': 'sched_getscheduler',
6212                 'reason': set([('task_struct', 'cred')])},
6213                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
6214                {'call': 'sched_getattr',
6215                 'reason': set([('task_struct', 'cred')])},
6216                {'call': 'getrusage',
6217                 'reason': set([('task_struct', 'cred')])},
6218                {'call': 'sched_setscheduler',
6219                 'reason': set([('task_struct', 'cred')])},
6220                {'call': 'setitimer',
6221                 'reason': set([('task_struct', 'cred')])},
6222                {'call': 'ioprio_get',
6223                 'reason': set([('task_struct', 'cred')])},
6224                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
6225                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
6226                {'call': 'move_pages',
6227                 'reason': set([('task_struct', 'cred')])},
6228                {'call': 'setpriority',
6229                 'reason': set([('task_struct', 'cred')])},
6230                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
6231                {'call': 'sched_getparam',
6232                 'reason': set([('task_struct', 'cred')])}],
6233  'getresgid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
6234                  {'call': 'rt_sigtimedwait',
6235                   'reason': set([('task_struct', 'cred')])},
6236                  {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
6237                  {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
6238                  {'call': 'sched_getaffinity',
6239                   'reason': set([('task_struct', 'cred')])},
6240                  {'call': 'sched_setparam',
6241                   'reason': set([('task_struct', 'cred')])},
6242                  {'call': 'ioprio_set',
6243                   'reason': set([('task_struct', 'cred')])},
6244                  {'call': 'getppid',
6245                   'reason': set([('task_struct', 'cred')])},
6246                  {'call': 'mq_timedreceive',
6247                   'reason': set([('task_struct', 'cred')])},
6248                  {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
6249                  {'call': 'sched_setaffinity',
6250                   'reason': set([('task_struct', 'cred')])},
6251                  {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
6252                  {'call': 'semtimedop',
6253                   'reason': set([('task_struct', 'cred')])},
6254                  {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
6255                  {'call': 'sched_rr_get_interval',
6256                   'reason': set([('task_struct', 'cred')])},
6257                  {'call': 'rt_sigprocmask',
6258                   'reason': set([('task_struct', 'cred')])},
6259                  {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
6260                  {'call': 'sigaltstack',
6261                   'reason': set([('task_struct', 'cred')])},
6262                  {'call': 'sched_setattr',
6263                   'reason': set([('task_struct', 'cred')])},
```



```
6264                {'call': 'migrate_pages',
6265                 'reason': set([('task_struct', 'cred')])},
6266                {'call': 'getitimer',
6267                 'reason': set([('task_struct', 'cred')])},
6268                {'call': 'setpgid',
6269                 'reason': set([('task_struct', 'cred')])},
6270                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
6271                {'call': 'prlimit64',
6272                 'reason': set([('task_struct', 'cred')])},
6273                {'call': 'perf_event_open',
6274                 'reason': set([('task_struct', 'cred')])},
6275                {'call': 'rt_sigaction',
6276                 'reason': set([('task_struct', 'cred')])},
6277                {'call': 'getpgid',
6278                 'reason': set([('task_struct', 'cred')])},
6279                {'call': 'getpriority',
6280                 'reason': set([('task_struct', 'cred')])},
6281                {'call': 'sigaction',
6282                 'reason': set([('task_struct', 'cred')])},
6283                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
6284                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
6285                {'call': 'get_robust_list',
6286                 'reason': set([('task_struct', 'cred')])},
6287                {'call': 'mq_timedsend',
6288                 'reason': set([('task_struct', 'cred')])},
6289                {'call': 'sched_getscheduler',
6290                 'reason': set([('task_struct', 'cred')])},
6291                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
6292                {'call': 'sched_getattr',
6293                 'reason': set([('task_struct', 'cred')])},
6294                {'call': 'getrusage',
6295                 'reason': set([('task_struct', 'cred')])},
6296                {'call': 'sched_setscheduler',
6297                 'reason': set([('task_struct', 'cred')])},
6298                {'call': 'setitimer',
6299                 'reason': set([('task_struct', 'cred')])},
6300                {'call': 'ioprio_get',
6301                 'reason': set([('task_struct', 'cred')])},
6302                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
6303                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
6304                {'call': 'move_pages',
6305                 'reason': set([('task_struct', 'cred')])},
6306                {'call': 'setpriority',
6307                 'reason': set([('task_struct', 'cred')])},
6308                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
6309                {'call': 'sched_getparam',
6310                 'reason': set([('task_struct', 'cred')])}],
6311  'getresuid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
6312                 {'call': 'rt_sigtimedwait',
6313                  'reason': set([('task_struct', 'cred')])},
6314                 {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
6315                 {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
6316                 {'call': 'sched_getaffinity',
6317                  'reason': set([('task_struct', 'cred')])},
6318                 {'call': 'sched_setparam',
6319                  'reason': set([('task_struct', 'cred')])},
6320                 {'call': 'ioprio_set',
6321                  'reason': set([('task_struct', 'cred')])},
6322                 {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
6323                 {'call': 'mq_timedreceive',
6324                  'reason': set([('task_struct', 'cred')])},
6325                 {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
6326                 {'call': 'sched_setaffinity',
6327                  'reason': set([('task_struct', 'cred')])},
6328                 {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
6329                 {'call': 'semtimedop',
```

```
6330                 'reason': set([('task_struct', 'cred')])},
6331                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
6332                {'call': 'sched_rr_get_interval',
6333                 'reason': set([('task_struct', 'cred')])},
6334                {'call': 'rt_sigprocmask',
6335                 'reason': set([('task_struct', 'cred')])},
6336                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
6337                {'call': 'sigaltstack',
6338                 'reason': set([('task_struct', 'cred')])},
6339                {'call': 'sched_setattr',
6340                 'reason': set([('task_struct', 'cred')])},
6341                {'call': 'migrate_pages',
6342                 'reason': set([('task_struct', 'cred')])},
6343                {'call': 'getitimer',
6344                 'reason': set([('task_struct', 'cred')])},
6345                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
6346                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
6347                {'call': 'prlimit64',
6348                 'reason': set([('task_struct', 'cred')])},
6349                {'call': 'perf_event_open',
6350                 'reason': set([('task_struct', 'cred')])},
6351                {'call': 'rt_sigaction',
6352                 'reason': set([('task_struct', 'cred')])},
6353                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
6354                {'call': 'getpriority',
6355                 'reason': set([('task_struct', 'cred')])},
6356                {'call': 'sigaction',
6357                 'reason': set([('task_struct', 'cred')])},
6358                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
6359                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
6360                {'call': 'get_robust_list',
6361                 'reason': set([('task_struct', 'cred')])},
6362                {'call': 'mq_timedsend',
6363                 'reason': set([('task_struct', 'cred')])},
6364                {'call': 'sched_getscheduler',
6365                 'reason': set([('task_struct', 'cred')])},
6366                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
6367                {'call': 'sched_getattr',
6368                 'reason': set([('task_struct', 'cred')])},
6369                {'call': 'getrusage',
6370                 'reason': set([('task_struct', 'cred')])},
6371                {'call': 'sched_setscheduler',
6372                 'reason': set([('task_struct', 'cred')])},
6373                {'call': 'setitimer',
6374                 'reason': set([('task_struct', 'cred')])},
6375                {'call': 'ioprio_get',
6376                 'reason': set([('task_struct', 'cred')])},
6377                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
6378                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
6379                {'call': 'move_pages',
6380                 'reason': set([('task_struct', 'cred')])},
6381                {'call': 'setpriority',
6382                 'reason': set([('task_struct', 'cred')])},
6383                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
6384                {'call': 'sched_getparam',
6385                 'reason': set([('task_struct', 'cred')])}],
6386  'getresuid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
6387                  {'call': 'rt_sigtimedwait',
6388                   'reason': set([('task_struct', 'cred')])},
6389                  {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
6390                  {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
6391                  {'call': 'sched_getaffinity',
6392                   'reason': set([('task_struct', 'cred')])},
6393                  {'call': 'sched_setparam',
6394                   'reason': set([('task_struct', 'cred')])},
6395                  {'call': 'ioprio_set',
```



```
6396                 'reason': set([('task_struct', 'cred')])},
6397                {'call': 'getppid',
6398                 'reason': set([('task_struct', 'cred')])},
6399                {'call': 'mq_timedreceive',
6400                 'reason': set([('task_struct', 'cred')])},
6401                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
6402                {'call': 'sched_setaffinity',
6403                 'reason': set([('task_struct', 'cred')])},
6404                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
6405                {'call': 'semtimedop',
6406                 'reason': set([('task_struct', 'cred')])},
6407                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
6408                {'call': 'sched_rr_get_interval',
6409                 'reason': set([('task_struct', 'cred')])},
6410                {'call': 'rt_sigprocmask',
6411                 'reason': set([('task_struct', 'cred')])},
6412                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
6413                {'call': 'sigaltstack',
6414                 'reason': set([('task_struct', 'cred')])},
6415                {'call': 'sched_setattr',
6416                 'reason': set([('task_struct', 'cred')])},
6417                {'call': 'migrate_pages',
6418                 'reason': set([('task_struct', 'cred')])},
6419                {'call': 'getitimer',
6420                 'reason': set([('task_struct', 'cred')])},
6421                {'call': 'setpgid',
6422                 'reason': set([('task_struct', 'cred')])},
6423                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
6424                {'call': 'prlimit64',
6425                 'reason': set([('task_struct', 'cred')])},
6426                {'call': 'perf_event_open',
6427                 'reason': set([('task_struct', 'cred')])},
6428                {'call': 'rt_sigaction',
6429                 'reason': set([('task_struct', 'cred')])},
6430                {'call': 'getpgid',
6431                 'reason': set([('task_struct', 'cred')])},
6432                {'call': 'getpriority',
6433                 'reason': set([('task_struct', 'cred')])},
6434                {'call': 'sigaction',
6435                 'reason': set([('task_struct', 'cred')])},
6436                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
6437                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
6438                {'call': 'get_robust_list',
6439                 'reason': set([('task_struct', 'cred')])},
6440                {'call': 'mq_timedsend',
6441                 'reason': set([('task_struct', 'cred')])},
6442                {'call': 'sched_getscheduler',
6443                 'reason': set([('task_struct', 'cred')])},
6444                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
6445                {'call': 'sched_getattr',
6446                 'reason': set([('task_struct', 'cred')])},
6447                {'call': 'getrusage',
6448                 'reason': set([('task_struct', 'cred')])},
6449                {'call': 'sched_setscheduler',
6450                 'reason': set([('task_struct', 'cred')])},
6451                {'call': 'setitimer',
6452                 'reason': set([('task_struct', 'cred')])},
6453                {'call': 'ioprio_get',
6454                 'reason': set([('task_struct', 'cred')])},
6455                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
6456                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
6457                {'call': 'move_pages',
6458                 'reason': set([('task_struct', 'cred')])},
6459                {'call': 'setpriority',
6460                 'reason': set([('task_struct', 'cred')])},
6461                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
```

```
6462                 {'call': 'sched_getparam',
6463                  'reason': set([('task_struct', 'cred')])}],
6464  'getrlimit': [{'call': 'setrlimit',
6465                 'reason': set([('rlimit', 'rlim_cur'),
6466                                ('rlimit', 'rlim_max')])},
6467                {'call': 'old_getrlimit',
6468                 'reason': set([('rlimit', 'rlim_cur'),
6469                                ('rlimit', 'rlim_max')])},
6470                {'call': 'prlimit64',
6471                 'reason': set([('rlimit', 'rlim_cur'),
6472                                ('rlimit', 'rlim_max')])}],
6473  'getrusage': [{'call': 'timer_create',
6474                 'reason': set([('signal_struct', 'maxrss')])},
6475                {'call': 'exit_group',
6476                 'reason': set([('signal_struct', 'maxrss')])}],
6477  'getsockname': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
6478                  {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
6479                  {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
6480                  {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
6481                  {'call': 'readahead', 'reason': set([('fd', 'file')])},
6482                  {'call': 'getdents', 'reason': set([('fd', 'file')])},
6483                  {'call': 'writev', 'reason': set([('fd', 'file')])},
6484                  {'call': 'preadv64', 'reason': set([('fd', 'file')])},
6485                  {'call': 'fchmod', 'reason': set([('fd', 'file')])},
6486                  {'call': 'pread64', 'reason': set([('fd', 'file')])},
6487                  {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
6488                  {'call': 'read', 'reason': set([('fd', 'file')])},
6489                  {'call': 'fchown', 'reason': set([('fd', 'file')])},
6490                  {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
6491                  {'call': 'utime', 'reason': set([('fd', 'file')])},
6492                  {'call': 'fsync', 'reason': set([('fd', 'file')])},
6493                  {'call': 'bpf', 'reason': set([('fd', 'file')])},
6494                  {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
6495                  {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
6496                  {'call': 'sendto', 'reason': set([('fd', 'file')])},
6497                  {'call': 'tee', 'reason': set([('fd', 'file')])},
6498                  {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
6499                  {'call': 'lseek', 'reason': set([('fd', 'file')])},
6500                  {'call': 'connect', 'reason': set([('fd', 'file')])},
6501                  {'call': 'getsockname', 'reason': set([('fd', 'file')])},
6502                  {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
6503                  {'call': 'flock', 'reason': set([('fd', 'file')])},
6504                  {'call': 'pwritev', 'reason': set([('fd', 'file')])},
6505                  {'call': 'fchdir', 'reason': set([('fd', 'file')])},
6506                  {'call': 'accept4', 'reason': set([('fd', 'file')])},
6507                  {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
6508                  {'call': 'inotify_rm_watch',
6509                   'reason': set([('fd', 'file')])},
6510                  {'call': 'utimensat', 'reason': set([('fd', 'file')])},
6511                  {'call': 'inotify_add_watch',
6512                   'reason': set([('fd', 'file')])},
6513                  {'call': 'preadv2', 'reason': set([('fd', 'file')])},
6514                  {'call': 'splice', 'reason': set([('fd', 'file')])},
6515                  {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
6516                  {'call': 'preadv', 'reason': set([('fd', 'file')])},
6517                  {'call': 'getpeername', 'reason': set([('fd', 'file')])},
6518                  {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
6519                  {'call': 'fcntl', 'reason': set([('fd', 'file')])},
6520                  {'call': 'ioctl', 'reason': set([('fd', 'file')])},
6521                  {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
6522                  {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
6523                  {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
6524                  {'call': 'futimesat', 'reason': set([('fd', 'file')])},
6525                  {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
6526                  {'call': 'shutdown', 'reason': set([('fd', 'file')])},
6527                  {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
```



```
6528                 {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
6529                 {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
6530                 {'call': 'getdents64', 'reason': set([('fd', 'file')])},
6531                 {'call': 'listen', 'reason': set([('fd', 'file')])},
6532                 {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
6533                 {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
6534                 {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
6535                 {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
6536                 {'call': 'fallocate', 'reason': set([('fd', 'file')])},
6537                 {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
6538                 {'call': 'llseek', 'reason': set([('fd', 'file')])},
6539                 {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
6540                 {'call': 'readv', 'reason': set([('fd', 'file')])},
6541                 {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
6542                 {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
6543                 {'call': 'write', 'reason': set([('fd', 'file')])},
6544                 {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
6545                 {'call': 'sendfile', 'reason': set([('fd', 'file')])},
6546                 {'call': 'bind', 'reason': set([('fd', 'file')])},
6547                 {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
6548                 {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
6549  'getsockopt': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
6550                 {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
6551                 {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
6552                 {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
6553                 {'call': 'readahead', 'reason': set([('fd', 'file')])},
6554                 {'call': 'getdents', 'reason': set([('fd', 'file')])},
6555                 {'call': 'writev', 'reason': set([('fd', 'file')])},
6556                 {'call': 'preadv64', 'reason': set([('fd', 'file')])},
6557                 {'call': 'fchmod', 'reason': set([('fd', 'file')])},
6558                 {'call': 'pread64', 'reason': set([('fd', 'file')])},
6559                 {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
6560                 {'call': 'read', 'reason': set([('fd', 'file')])},
6561                 {'call': 'fchown', 'reason': set([('fd', 'file')])},
6562                 {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
6563                 {'call': 'utime', 'reason': set([('fd', 'file')])},
6564                 {'call': 'fsync', 'reason': set([('fd', 'file')])},
6565                 {'call': 'bpf', 'reason': set([('fd', 'file')])},
6566                 {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
6567                 {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
6568                 {'call': 'sendto', 'reason': set([('fd', 'file')])},
6569                 {'call': 'tee', 'reason': set([('fd', 'file')])},
6570                 {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
6571                 {'call': 'lseek', 'reason': set([('fd', 'file')])},
6572                 {'call': 'connect', 'reason': set([('fd', 'file')])},
6573                 {'call': 'getsockname', 'reason': set([('fd', 'file')])},
6574                 {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
6575                 {'call': 'flock', 'reason': set([('fd', 'file')])},
6576                 {'call': 'pwritev', 'reason': set([('fd', 'file')])},
6577                 {'call': 'fchdir', 'reason': set([('fd', 'file')])},
6578                 {'call': 'accept4',
6579                  'reason': set([('fd', 'file'),
6580                                 ('proto_ops', 'compat_getsockopt')])},
6581                 {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
6582                 {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
6583                 {'call': 'utimensat', 'reason': set([('fd', 'file')])},
6584                 {'call': 'inotify_add_watch',
6585                  'reason': set([('fd', 'file')])},
6586                 {'call': 'preadv2', 'reason': set([('fd', 'file')])},
6587                 {'call': 'splice', 'reason': set([('fd', 'file')])},
6588                 {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
6589                 {'call': 'preadv', 'reason': set([('fd', 'file')])},
6590                 {'call': 'getpeername', 'reason': set([('fd', 'file')])},
6591                 {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
6592                 {'call': 'fcntl', 'reason': set([('fd', 'file')])},
6593                 {'call': 'ioctl', 'reason': set([('fd', 'file')])},
```

```
                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
                {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
                {'call': 'futimesat', 'reason': set([('fd', 'file')])},
                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
                {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
                {'call': 'listen', 'reason': set([('fd', 'file')])},
                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
                {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
                {'call': 'llseek', 'reason': set([('fd', 'file')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
                {'call': 'readv', 'reason': set([('fd', 'file')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
                {'call': 'write', 'reason': set([('fd', 'file')])},
                {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
                {'call': 'sendfile', 'reason': set([('fd', 'file')])},
                {'call': 'bind', 'reason': set([('fd', 'file')])},
                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
 'getuid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
            {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setaffinity',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
            {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
            {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
            {'call': 'perf_event_open',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
```



```
            {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
            {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
            {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
            {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
            {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
            {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
            {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getparam',
             'reason': set([('task_struct', 'cred')])}],
 'getuid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_getaffinity',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setparam',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'ioprio_set',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'mq_timedreceive',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setaffinity',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
              {'call': 'semtimedop',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sigaltstack',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setattr',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'migrate_pages',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
              {'call': 'perf_event_open',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'rt_sigaction',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
```

```
6726                {'call': 'getpriority',
6727                 'reason': set([('task_struct', 'cred')])},
6728                {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
6729                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
6730                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
6731                {'call': 'get_robust_list',
6732                 'reason': set([('task_struct', 'cred')])},
6733                {'call': 'mq_timedsend',
6734                 'reason': set([('task_struct', 'cred')])},
6735                {'call': 'sched_getscheduler',
6736                 'reason': set([('task_struct', 'cred')])},
6737                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
6738                {'call': 'sched_getattr',
6739                 'reason': set([('task_struct', 'cred')])},
6740                {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
6741                {'call': 'sched_setscheduler',
6742                 'reason': set([('task_struct', 'cred')])},
6743                {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
6744                {'call': 'ioprio_get',
6745                 'reason': set([('task_struct', 'cred')])},
6746                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
6747                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
6748                {'call': 'move_pages',
6749                 'reason': set([('task_struct', 'cred')])},
6750                {'call': 'setpriority',
6751                 'reason': set([('task_struct', 'cred')])},
6752                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
6753                {'call': 'sched_getparam',
6754                 'reason': set([('task_struct', 'cred')])}],
6755  'getxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
6756                {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
6757                {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
6758                {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
6759                {'call': 'remap_file_pages',
6760                 'reason': set([('path', 'dentry')])},
6761                {'call': 'dup3', 'reason': set([('path', 'dentry')])},
6762                {'call': 'unshare', 'reason': set([('path', 'dentry')])},
6763                {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
6764                {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
6765                {'call': 'flock', 'reason': set([('path', 'dentry')])},
6766                {'call': 'openat', 'reason': set([('path', 'dentry')])},
6767                {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
6768                {'call': 'uselib', 'reason': set([('path', 'dentry')])},
6769                {'call': 'accept4', 'reason': set([('path', 'dentry')])},
6770                {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
6771                {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
6772                {'call': 'shmat', 'reason': set([('path', 'dentry')])},
6773                {'call': 'socket', 'reason': set([('path', 'dentry')])},
6774                {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
6775                {'call': 'perf_event_open',
6776                 'reason': set([('path', 'dentry')])},
6777                {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
6778                {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
6779                {'call': 'acct', 'reason': set([('path', 'dentry')])},
6780                {'call': 'open', 'reason': set([('path', 'dentry')])},
6781                {'call': 'dup', 'reason': set([('path', 'dentry')])},
6782                {'call': 'setns', 'reason': set([('path', 'dentry')])},
6783                {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
6784                {'call': 'swapon', 'reason': set([('path', 'dentry')])},
6785                {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
6786                {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
6787                {'call': 'open_by_handle_at',
6788                 'reason': set([('path', 'dentry')])}],
6789  'init_module': [{'call': 'delete_module',
6790                   'reason': set([('module', 'args'),
6791                                  ('module', 'kp'),
```



```
6792                                  ('module', 'num_kp'),
6793                                  ('module_layout', 'base'),
6794                                  ('module_layout', 'size')])},
6795                  {'call': 'init_module',
6796                   'reason': set([('load_info', 'debug'),
6797                                  ('load_info', 'hdr'),
6798                                  ('load_info', 'len'),
6799                                  ('load_info', 'num_debug'),
6800                                  ('module', 'args'),
6801                                  ('module', 'kp'),
6802                                  ('module', 'num_kp'),
6803                                  ('module_layout', 'base'),
6804                                  ('module_layout', 'size')])},
6805                  {'call': 'finit_module',
6806                   'reason': set([('load_info', 'debug'),
6807                                  ('load_info', 'hdr'),
6808                                  ('load_info', 'len'),
6809                                  ('load_info', 'num_debug'),
6810                                  ('module', 'args'),
6811                                  ('module', 'kp'),
6812                                  ('module', 'num_kp'),
6813                                  ('module_layout', 'base'),
6814                                  ('module_layout', 'size')])}],
6815  'inotify_add_watch': [{'call': 'syncfs',
6816                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6817                        {'call': 'vmsplice',
6818                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6819                        {'call': 'eventfd2',
6820                         'reason': set([('file', 'f_op')])},
6821                        {'call': 'mq_unlink',
6822                         'reason': set([('dentry', 'd_inode')])},
6823                        {'call': 'pwritev64',
6824                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6825                        {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
6826                        {'call': 'fremovexattr',
6827                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6828                        {'call': 'readahead',
6829                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6830                        {'call': 'getdents',
6831                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6832                        {'call': 'writev',
6833                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6834                        {'call': 'preadv64',
6835                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6836                        {'call': 'fchmod',
6837                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6838                        {'call': 'pread64',
6839                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6840                        {'call': 'pivot_root',
6841                         'reason': set([('dentry', 'd_inode')])},
6842                        {'call': 'signalfd4',
6843                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6844                        {'call': 'memfd_create',
6845                         'reason': set([('file', 'f_op')])},
6846                        {'call': 'remap_file_pages',
6847                         'reason': set([('file', 'f_op')])},
6848                        {'call': 'dup3', 'reason': set([('file', 'f_op')])},
6849                        {'call': 'read',
6850                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6851                        {'call': 'fchown',
6852                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6853                        {'call': 'mq_timedreceive',
6854                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6855                        {'call': 'utime',
6856                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
6857                        {'call': 'fsync',
```

```
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'bpf',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'recvfrom',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fsetxattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'sendto',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mkdirat',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'epoll_create1',
                     'reason': set([('file', 'f_op')])},
                    {'call': 'tee',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'sync_file_range',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'lseek',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'connect',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'getsockname',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'epoll_ctl',
                     'reason': set([('fd', 'file'),
                                    ('fd', 'flags'),
                                    ('file', 'f_op')])},
                    {'call': 'flock',
                     'reason': set([('fd', 'file'),
                                    ('fd', 'flags'),
                                    ('file', 'f_op')])},
                    {'call': 'pwritev',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fchdir',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'openat', 'reason': set([('file', 'f_op')])},
                    {'call': 'uselib', 'reason': set([('file', 'f_op')])},
                    {'call': 'renameat2',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'accept4',
                     'reason': set([('fd', 'file'),
                                    ('fd', 'flags'),
                                    ('file', 'f_op')])},
                    {'call': 'old_readdir',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'inotify_rm_watch',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'socketpair',
                     'reason': set([('file', 'f_op')])},
                    {'call': 'utimensat',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'getcwd',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'inotify_add_watch',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'preadv2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'splice',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'ftruncate',
                     'reason': set([('dentry', 'd_inode'),
                                    ('fd', 'file'),
                                    ('fd', 'flags')])},
                    {'call': 'preadv',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'getpeername',
```



```
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'shmat', 'reason': set([('file', 'f_op')])},
                    {'call': 'setsockopt',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mknodat',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'socket', 'reason': set([('file', 'f_op')])},
                    {'call': 'symlinkat',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
                    {'call': 'fcntl',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'ioctl',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'pwrite64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'perf_event_open',
                     'reason': set([('fd', 'file'),
                                    ('fd', 'flags'),
                                    ('file', 'f_op')])},
                    {'call': 'linkat',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
                    {'call': 'pwritev64v2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'futimesat',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'pwritev2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'shutdown',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'acct', 'reason': set([('file', 'f_op')])},
                    {'call': 'open', 'reason': set([('file', 'f_op')])},
                    {'call': 'unlink',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'getsockopt',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_getsetattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'rmdir',
                     'reason': set([('dentry', 'd_inode')])},
                    {'call': 'dup', 'reason': set([('file', 'f_op')])},
                    {'call': 'fdatasync',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'setns', 'reason': set([('file', 'f_op')])},
                    {'call': 'getdents64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'listen',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'copy_file_range',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fgetxattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
                    {'call': 'fcntl64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'swapon', 'reason': set([('file', 'f_op')])},
                    {'call': 'fallocate',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'epoll_wait',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'llseek',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mmap_pgoff',
```

```
                        'reason': set([('file', 'f_op')])},
                       {'call': 'preadv64v2',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'readv',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'fstatfs',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'fstatfs64',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'write',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'mq_notify',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'sendfile',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'mq_open',
                        'reason': set([('dentry', 'd_inode'),
                                       ('file', 'f_op')])},
                       {'call': 'unlinkat',
                        'reason': set([('dentry', 'd_inode')])},
                       {'call': 'open_by_handle_at',
                        'reason': set([('file', 'f_op')])},
                       {'call': 'bind',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'flistxattr',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])},
                       {'call': 'sendfile64',
                        'reason': set([('fd', 'file'), ('fd', 'flags')])}],
 'inotify_init1': [{'call': 'keyctl',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'msgrcv',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setparam',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'ioprio_set',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getppid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'capget',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setaffinity',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'signal',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'semtimedop',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'umount',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_rr_get_interval',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'rt_sigprocmask',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setsid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sigaltstack',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setattr',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'inotify_rm_watch',
```



```
                    'reason': set([('fsnotify_group', 'overflow_event'),
                                   ('inotify_group_private_data',
                                    'ucounts')])},
                   {'call': 'migrate_pages',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getitimer',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setpgid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'inotify_add_watch',
                    'reason': set([('fsnotify_group', 'overflow_event'),
                                   ('inotify_group_private_data',
                                    'ucounts')])},
                   {'call': 'getsid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'prlimit64',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'perf_event_open',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'rt_sigaction',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getpgid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getpriority',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sigaction',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setns',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                   {'call': 'get_robust_list',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'mq_timedsend',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getscheduler',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'ptrace',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getattr',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getrusage',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setscheduler',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setitimer',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'ioprio_get',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'vfork',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'prctl',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'move_pages',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setpriority',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'inotify_init1',
                    'reason': set([('fsnotify_group', 'overflow_event'),
                                   ('inotify_group_private_data',
                                    'ucounts')])},
                   {'call': 'clone',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getparam',
                    'reason': set([('task_struct', 'cred')])}],
 'inotify_rm_watch': [{'call': 'syncfs',
                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
7122                    {'call': 'vmsplice',
7123                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7124                    {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
7125                    {'call': 'pwritev64',
7126                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7127                    {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
7128                    {'call': 'fremovexattr',
7129                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7130                    {'call': 'readahead',
7131                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7132                    {'call': 'getdents',
7133                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7134                    {'call': 'writev',
7135                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7136                    {'call': 'preadv64',
7137                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7138                    {'call': 'fchmod',
7139                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7140                    {'call': 'pread64',
7141                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7142                    {'call': 'signalfd4',
7143                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7144                    {'call': 'memfd_create',
7145                     'reason': set([('file', 'f_op')])},
7146                    {'call': 'remap_file_pages',
7147                     'reason': set([('file', 'f_op')])},
7148                    {'call': 'dup3', 'reason': set([('file', 'f_op')])},
7149                    {'call': 'read',
7150                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7151                    {'call': 'fchown',
7152                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7153                    {'call': 'mq_timedreceive',
7154                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7155                    {'call': 'utime',
7156                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7157                    {'call': 'fsync',
7158                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7159                    {'call': 'bpf',
7160                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7161                    {'call': 'recvfrom',
7162                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7163                    {'call': 'fsetxattr',
7164                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7165                    {'call': 'sendto',
7166                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7167                    {'call': 'epoll_create1',
7168                     'reason': set([('file', 'f_op')])},
7169                    {'call': 'tee',
7170                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7171                    {'call': 'sync_file_range',
7172                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7173                    {'call': 'lseek',
7174                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7175                    {'call': 'connect',
7176                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7177                    {'call': 'getsockname',
7178                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7179                    {'call': 'epoll_ctl',
7180                     'reason': set([('fd', 'file'),
7181                                    ('fd', 'flags'),
7182                                    ('file', 'f_op')])},
7183                    {'call': 'flock',
7184                     'reason': set([('fd', 'file'),
7185                                    ('fd', 'flags'),
7186                                    ('file', 'f_op')])},
7187                    {'call': 'pwritev',
```



```
7188                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7189                    {'call': 'fchdir',
7190                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7191                    {'call': 'openat', 'reason': set([('file', 'f_op')])},
7192                    {'call': 'uselib', 'reason': set([('file', 'f_op')])},
7193                    {'call': 'accept4',
7194                     'reason': set([('fd', 'file'),
7195                                    ('fd', 'flags'),
7196                                    ('file', 'f_op')])},
7197                    {'call': 'old_readdir',
7198                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7199                    {'call': 'inotify_rm_watch',
7200                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7201                    {'call': 'socketpair',
7202                     'reason': set([('file', 'f_op')])},
7203                    {'call': 'utimensat',
7204                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7205                    {'call': 'inotify_add_watch',
7206                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7207                    {'call': 'preadv2',
7208                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7209                    {'call': 'splice',
7210                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7211                    {'call': 'ftruncate',
7212                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7213                    {'call': 'preadv',
7214                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7215                    {'call': 'getpeername',
7216                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7217                    {'call': 'shmat', 'reason': set([('file', 'f_op')])},
7218                    {'call': 'setsockopt',
7219                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7220                    {'call': 'socket', 'reason': set([('file', 'f_op')])},
7221                    {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
7222                    {'call': 'fcntl',
7223                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7224                    {'call': 'ioctl',
7225                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7226                    {'call': 'pwrite64',
7227                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7228                    {'call': 'perf_event_open',
7229                     'reason': set([('fd', 'file'),
7230                                    ('fd', 'flags'),
7231                                    ('file', 'f_op')])},
7232                    {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
7233                    {'call': 'pwritev64v2',
7234                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7235                    {'call': 'futimesat',
7236                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7237                    {'call': 'pwritev2',
7238                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7239                    {'call': 'shutdown',
7240                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7241                    {'call': 'acct', 'reason': set([('file', 'f_op')])},
7242                    {'call': 'open', 'reason': set([('file', 'f_op')])},
7243                    {'call': 'getsockopt',
7244                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7245                    {'call': 'mq_getsetattr',
7246                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7247                    {'call': 'dup', 'reason': set([('file', 'f_op')])},
7248                    {'call': 'fdatasync',
7249                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7250                    {'call': 'setns', 'reason': set([('file', 'f_op')])},
7251                    {'call': 'getdents64',
7252                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
7253                    {'call': 'listen',
```

```
7254                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7255                      {'call': 'copy_file_range',
7256                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7257                      {'call': 'mq_timedsend',
7258                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7259                      {'call': 'fgetxattr',
7260                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7261                      {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
7262                      {'call': 'fcntl64',
7263                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7264                      {'call': 'swapon', 'reason': set([('file', 'f_op')])},
7265                      {'call': 'fallocate',
7266                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7267                      {'call': 'epoll_wait',
7268                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7269                      {'call': 'llseek',
7270                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7271                      {'call': 'mmap_pgoff',
7272                       'reason': set([('file', 'f_op')])},
7273                      {'call': 'preadv64v2',
7274                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7275                      {'call': 'readv',
7276                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7277                      {'call': 'fstatfs',
7278                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7279                      {'call': 'fstatfs64',
7280                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7281                      {'call': 'write',
7282                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7283                      {'call': 'mq_notify',
7284                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7285                      {'call': 'sendfile',
7286                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7287                      {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
7288                      {'call': 'open_by_handle_at',
7289                       'reason': set([('file', 'f_op')])},
7290                      {'call': 'bind',
7291                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7292                      {'call': 'flistxattr',
7293                       'reason': set([('fd', 'file'), ('fd', 'flags')])},
7294                      {'call': 'sendfile64',
7295                       'reason': set([('fd', 'file'), ('fd', 'flags')])}],
7296   'io_cancel': [{'call': 'swapoff',
7297                  'reason': set([('mm_struct', 'ioctx_table')])},
7298                 {'call': 'remap_file_pages',
7299                  'reason': set([('mm_struct', 'ioctx_table')])},
7300                 {'call': 'io_getevents',
7301                  'reason': set([('kioctx', 'user_id'),
7302                                 ('kioctx_table', 'nr'),
7303                                 ('mm_struct', 'ioctx_table')])},
7304                 {'call': 'migrate_pages',
7305                  'reason': set([('mm_struct', 'ioctx_table')])},
7306                 {'call': 'shmdt',
7307                  'reason': set([('mm_struct', 'ioctx_table')])},
7308                 {'call': 'brk', 'reason': set([('mm_struct', 'ioctx_table')])},
7309                 {'call': 'get_mempolicy',
7310                  'reason': set([('mm_struct', 'ioctx_table')])},
7311                 {'call': 'io_submit', 'reason': set([('kioctx', 'user_id')])},
7312                 {'call': 'getrusage',
7313                  'reason': set([('mm_struct', 'ioctx_table')])},
7314                 {'call': 'io_setup',
7315                  'reason': set([('kioctx', 'user_id'),
7316                                 ('kioctx_table', 'nr'),
7317                                 ('mm_struct', 'ioctx_table')])},
7318                 {'call': 'mremap',
7319                  'reason': set([('mm_struct', 'ioctx_table')])},
```



```
7320                 {'call': 'io_destroy',
7321                  'reason': set([('kioctx', 'user_id'),
7322                                 ('kioctx_table', 'nr'),
7323                                 ('mm_struct', 'ioctx_table')])},
7324                 {'call': 'mbind',
7325                  'reason': set([('mm_struct', 'ioctx_table')])},
7326                 {'call': 'prctl',
7327                  'reason': set([('mm_struct', 'ioctx_table')])},
7328                 {'call': 'move_pages',
7329                  'reason': set([('mm_struct', 'ioctx_table')])},
7330                 {'call': 'modify_ldt',
7331                  'reason': set([('mm_struct', 'ioctx_table')])},
7332                 {'call': 'mincore',
7333                  'reason': set([('mm_struct', 'ioctx_table')])},
7334                 {'call': 'io_cancel',
7335                  'reason': set([('aio_kiocb', 'ki_user_iocb'),
7336                                 ('kioctx', 'user_id'),
7337                                 ('kioctx_table', 'nr'),
7338                                 ('mm_struct', 'ioctx_table')])}],
7339   'io_destroy': [{'call': 'swapoff',
7340                   'reason': set([('mm_struct', 'ioctx_table')])},
7341                  {'call': 'remap_file_pages',
7342                   'reason': set([('mm_struct', 'ioctx_table')])},
7343                  {'call': 'io_getevents',
7344                   'reason': set([('kioctx', 'max_reqs'),
7345                                  ('kioctx', 'mmap_base'),
7346                                  ('kioctx', 'mmap_size'),
7347                                  ('kioctx', 'user_id'),
7348                                  ('kioctx_table', 'nr'),
7349                                  ('mm_struct', 'ioctx_table')])},
7350                  {'call': 'migrate_pages',
7351                   'reason': set([('mm_struct', 'ioctx_table')])},
7352                  {'call': 'shmdt',
7353                   'reason': set([('mm_struct', 'ioctx_table')])},
7354                  {'call': 'brk',
7355                   'reason': set([('mm_struct', 'ioctx_table')])},
7356                  {'call': 'get_mempolicy',
7357                   'reason': set([('mm_struct', 'ioctx_table')])},
7358                  {'call': 'io_submit',
7359                   'reason': set([('kioctx', 'max_reqs'),
7360                                  ('kioctx', 'mmap_base'),
7361                                  ('kioctx', 'mmap_size'),
7362                                  ('kioctx', 'user_id')])},
7363                  {'call': 'getrusage',
7364                   'reason': set([('mm_struct', 'ioctx_table')])},
7365                  {'call': 'io_setup',
7366                   'reason': set([('kioctx', 'max_reqs'),
7367                                  ('kioctx', 'mmap_base'),
7368                                  ('kioctx', 'mmap_size'),
7369                                  ('kioctx', 'user_id'),
7370                                  ('kioctx_table', 'nr'),
7371                                  ('mm_struct', 'ioctx_table')])},
7372                  {'call': 'mremap',
7373                   'reason': set([('mm_struct', 'ioctx_table')])},
7374                  {'call': 'io_destroy',
7375                   'reason': set([('kioctx', 'max_reqs'),
7376                                  ('kioctx', 'mmap_base'),
7377                                  ('kioctx', 'mmap_size'),
7378                                  ('kioctx', 'user_id'),
7379                                  ('kioctx_table', 'nr'),
7380                                  ('mm_struct', 'ioctx_table')])},
7381                  {'call': 'mbind',
7382                   'reason': set([('mm_struct', 'ioctx_table')])},
7383                  {'call': 'prctl',
7384                   'reason': set([('mm_struct', 'ioctx_table')])},
7385                  {'call': 'move_pages',
```

```
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'modify_ldt',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'mincore',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'io_cancel',
                 'reason': set([('kioctx', 'max_reqs'),
                                ('kioctx', 'mmap_base'),
                                ('kioctx', 'mmap_size'),
                                ('kioctx', 'user_id'),
                                ('kioctx_table', 'nr'),
                                ('mm_struct', 'ioctx_table')])}],
 'io_getevents': [{'call': 'keyctl',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'rt_sigtimedwait',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'msgrcv',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'kill',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'swapoff',
                   'reason': set([('mm_struct', 'ioctx_table')])},
                  {'call': 'sched_getaffinity',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setparam',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'ioprio_set',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'remap_file_pages',
                   'reason': set([('mm_struct', 'ioctx_table')])},
                  {'call': 'io_getevents',
                   'reason': set([('kioctx', 'user_id'),
                                  ('kioctx_table', 'nr'),
                                  ('mm_struct', 'ioctx_table')])},
                  {'call': 'getppid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'mq_timedreceive',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'capget',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setaffinity',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'signal',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'semtimedop',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'umount',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_rr_get_interval',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'rt_sigprocmask',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setsid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sigaltstack',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setattr',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'migrate_pages',
                   'reason': set([('mm_struct', 'ioctx_table'),
                                  ('task_struct', 'timer_slack_ns')])},
                  {'call': 'getitimer',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setpgid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getsid',
```



```
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'prlimit64',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'perf_event_open',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'shmdt',
                   'reason': set([('mm_struct', 'ioctx_table')])},
                  {'call': 'rt_sigaction',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getpgid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'brk',
                   'reason': set([('mm_struct', 'ioctx_table')])},
                  {'call': 'getpriority',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sigaction',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setns',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'fork',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'get_mempolicy',
                   'reason': set([('mm_struct', 'ioctx_table')])},
                  {'call': 'io_submit',
                   'reason': set([('kioctx', 'user_id')])},
                  {'call': 'get_robust_list',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'mq_timedsend',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_getscheduler',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'ptrace',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_getattr',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getrusage',
                   'reason': set([('mm_struct', 'ioctx_table'),
                                  ('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setscheduler',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setitimer',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'ioprio_get',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'vfork',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'io_setup',
                   'reason': set([('kioctx', 'user_id'),
                                  ('kioctx_table', 'nr'),
                                  ('mm_struct', 'ioctx_table')])},
                  {'call': 'mremap',
                   'reason': set([('mm_struct', 'ioctx_table')])},
                  {'call': 'io_destroy',
                   'reason': set([('kioctx', 'user_id'),
                                  ('kioctx_table', 'nr'),
                                  ('mm_struct', 'ioctx_table')])},
                  {'call': 'mbind',
                   'reason': set([('mm_struct', 'ioctx_table')])},
                  {'call': 'prctl',
                   'reason': set([('mm_struct', 'ioctx_table'),
                                  ('task_struct', 'timer_slack_ns')])},
                  {'call': 'move_pages',
                   'reason': set([('mm_struct', 'ioctx_table'),
                                  ('task_struct', 'timer_slack_ns')])},
                  {'call': 'modify_ldt',
                   'reason': set([('mm_struct', 'ioctx_table')])},
```

```
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'mincore',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'io_cancel',
                 'reason': set([('kioctx', 'user_id'),
                                ('kioctx_table', 'nr'),
                                ('mm_struct', 'ioctx_table')])}],
 'io_setup': [{'call': 'io_getevents',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_submit',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_setup',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_destroy',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_cancel',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])}],
 'io_submit': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setparam',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioprio_set',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedreceive',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'semtimedop',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_rr_get_interval',
                'reason': set([('mm_segment_t', 'seg')])},
```



```
               {'call': 'rt_sigprocmask',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sigaltstack',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setattr',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'migrate_pages',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getitimer',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'prlimit64',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'perf_event_open',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigaction',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getpriority',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sigaction',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'get_robust_list',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedsend',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getscheduler',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getattr',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getrusage',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setscheduler',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setitimer',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioprio_get',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'move_pages',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setpriority',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getparam',
                'reason': set([('mm_segment_t', 'seg')])}],
 'ioctl': [{'call': 'syncfs',
            'reason': set([('fd', 'file'), ('fd', 'flags')])},
           {'call': 'vmsplice',
            'reason': set([('fd', 'file'), ('fd', 'flags')])},
           {'call': 'pwritev64',
            'reason': set([('fd', 'file'), ('fd', 'flags')])},
           {'call': 'fremovexattr',
            'reason': set([('fd', 'file'), ('fd', 'flags')])},
           {'call': 'readahead',
            'reason': set([('fd', 'file'), ('fd', 'flags')])},
           {'call': 'getdents',
            'reason': set([('fd', 'file'), ('fd', 'flags')])},
           {'call': 'writev',
            'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
7650            {'call': 'preadv64',
7651             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7652            {'call': 'fchmod',
7653             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7654            {'call': 'pread64',
7655             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7656            {'call': 'signalfd4',
7657             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7658            {'call': 'read', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
7659            {'call': 'fchown',
7660             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7661            {'call': 'mq_timedreceive',
7662             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7663            {'call': 'utime',
7664             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7665            {'call': 'fsync',
7666             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7667            {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
7668            {'call': 'recvfrom',
7669             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7670            {'call': 'fsetxattr',
7671             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7672            {'call': 'sendto',
7673             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7674            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
7675            {'call': 'sync_file_range',
7676             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7677            {'call': 'lseek',
7678             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7679            {'call': 'connect',
7680             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7681            {'call': 'getsockname',
7682             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7683            {'call': 'epoll_ctl',
7684             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7685            {'call': 'flock',
7686             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7687            {'call': 'pwritev',
7688             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7689            {'call': 'fchdir',
7690             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7691            {'call': 'accept4',
7692             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7693            {'call': 'old_readdir',
7694             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7695            {'call': 'inotify_rm_watch',
7696             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7697            {'call': 'utimensat',
7698             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7699            {'call': 'inotify_add_watch',
7700             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7701            {'call': 'preadv2',
7702             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7703            {'call': 'splice',
7704             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7705            {'call': 'ftruncate',
7706             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7707            {'call': 'preadv',
7708             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7709            {'call': 'getpeername',
7710             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7711            {'call': 'setsockopt',
7712             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7713            {'call': 'fcntl',
7714             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7715            {'call': 'ioctl',
```



```
7716             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7717            {'call': 'pwrite64',
7718             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7719            {'call': 'perf_event_open',
7720             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7721            {'call': 'pwritev64v2',
7722             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7723            {'call': 'futimesat',
7724             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7725            {'call': 'pwritev2',
7726             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7727            {'call': 'shutdown',
7728             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7729            {'call': 'getsockopt',
7730             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7731            {'call': 'mq_getsetattr',
7732             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7733            {'call': 'fdatasync',
7734             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7735            {'call': 'getdents64',
7736             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7737            {'call': 'listen',
7738             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7739            {'call': 'copy_file_range',
7740             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7741            {'call': 'mq_timedsend',
7742             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7743            {'call': 'fgetxattr',
7744             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7745            {'call': 'fcntl64',
7746             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7747            {'call': 'fallocate',
7748             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7749            {'call': 'epoll_wait',
7750             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7751            {'call': 'llseek',
7752             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7753            {'call': 'preadv64v2',
7754             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7755            {'call': 'readv',
7756             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7757            {'call': 'fstatfs',
7758             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7759            {'call': 'fstatfs64',
7760             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7761            {'call': 'write',
7762             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7763            {'call': 'mq_notify',
7764             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7765            {'call': 'sendfile',
7766             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7767            {'call': 'bind', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
7768            {'call': 'flistxattr',
7769             'reason': set([('fd', 'file'), ('fd', 'flags')])},
7770            {'call': 'sendfile64',
7771             'reason': set([('fd', 'file'), ('fd', 'flags')])}],
7772  'ioperm': [{'call': 'keyctl',
7773              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7774             {'call': 'rt_sigtimedwait',
7775              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7776             {'call': 'msgrcv',
7777              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7778             {'call': 'kill',
7779              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7780             {'call': 'sched_getaffinity',
7781              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
```

```
7782             {'call': 'sched_setparam',
7783              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7784             {'call': 'ioprio_set',
7785              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7786             {'call': 'getppid',
7787              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7788             {'call': 'ioperm',
7789              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7790             {'call': 'mq_timedreceive',
7791              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7792             {'call': 'capget',
7793              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7794             {'call': 'sched_setaffinity',
7795              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7796             {'call': 'signal',
7797              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7798             {'call': 'semtimedop',
7799              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7800             {'call': 'umount',
7801              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7802             {'call': 'sched_rr_get_interval',
7803              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7804             {'call': 'rt_sigprocmask',
7805              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7806             {'call': 'setsid',
7807              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7808             {'call': 'sigaltstack',
7809              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7810             {'call': 'sched_setattr',
7811              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7812             {'call': 'migrate_pages',
7813              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7814             {'call': 'getitimer',
7815              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7816             {'call': 'setpgid',
7817              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7818             {'call': 'getsid',
7819              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7820             {'call': 'prlimit64',
7821              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7822             {'call': 'perf_event_open',
7823              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7824             {'call': 'rt_sigaction',
7825              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7826             {'call': 'getpgid',
7827              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7828             {'call': 'getpriority',
7829              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7830             {'call': 'sigaction',
7831              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7832             {'call': 'setns',
7833              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7834             {'call': 'fork',
7835              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7836             {'call': 'get_robust_list',
7837              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7838             {'call': 'mq_timedsend',
7839              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7840             {'call': 'sched_getscheduler',
7841              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7842             {'call': 'ptrace',
7843              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7844             {'call': 'sched_getattr',
7845              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7846             {'call': 'getrusage',
7847              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
```



```
7848             {'call': 'sched_setscheduler',
7849              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7850             {'call': 'setitimer',
7851              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7852             {'call': 'ioprio_get',
7853              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7854             {'call': 'vfork',
7855              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7856             {'call': 'prctl',
7857              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7858             {'call': 'move_pages',
7859              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7860             {'call': 'setpriority',
7861              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7862             {'call': 'clone',
7863              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
7864             {'call': 'sched_getparam',
7865              'reason': set([('thread_struct', 'io_bitmap_ptr')])}],
7866  'ioprio_get': [{'call': 'keyctl',
7867                  'reason': set([('task_struct', 'cred'),
7868                                 ('task_struct', 'io_context'),
7869                                 ('task_struct', 'real_cred')])},
7870                 {'call': 'rt_sigtimedwait',
7871                  'reason': set([('task_struct', 'cred'),
7872                                 ('task_struct', 'io_context'),
7873                                 ('task_struct', 'real_cred')])},
7874                 {'call': 'msgrcv',
7875                  'reason': set([('task_struct', 'cred'),
7876                                 ('task_struct', 'io_context'),
7877                                 ('task_struct', 'real_cred')])},
7878                 {'call': 'kill',
7879                  'reason': set([('task_struct', 'cred'),
7880                                 ('task_struct', 'io_context'),
7881                                 ('task_struct', 'real_cred')])},
7882                 {'call': 'sched_getaffinity',
7883                  'reason': set([('task_struct', 'cred'),
7884                                 ('task_struct', 'io_context'),
7885                                 ('task_struct', 'real_cred')])},
7886                 {'call': 'sched_setparam',
7887                  'reason': set([('task_struct', 'cred'),
7888                                 ('task_struct', 'io_context'),
7889                                 ('task_struct', 'real_cred')])},
7890                 {'call': 'ioprio_set',
7891                  'reason': set([('task_struct', 'cred'),
7892                                 ('task_struct', 'io_context'),
7893                                 ('task_struct', 'real_cred')])},
7894                 {'call': 'getppid',
7895                  'reason': set([('task_struct', 'cred'),
7896                                 ('task_struct', 'io_context'),
7897                                 ('task_struct', 'real_cred')])},
7898                 {'call': 'mq_timedreceive',
7899                  'reason': set([('task_struct', 'cred'),
7900                                 ('task_struct', 'io_context'),
7901                                 ('task_struct', 'real_cred')])},
7902                 {'call': 'capget',
7903                  'reason': set([('task_struct', 'cred'),
7904                                 ('task_struct', 'io_context'),
7905                                 ('task_struct', 'real_cred')])},
7906                 {'call': 'sched_setaffinity',
7907                  'reason': set([('task_struct', 'cred'),
7908                                 ('task_struct', 'io_context'),
7909                                 ('task_struct', 'real_cred')])},
7910                 {'call': 'signal',
7911                  'reason': set([('task_struct', 'cred'),
7912                                 ('task_struct', 'io_context'),
7913                                 ('task_struct', 'real_cred')])},
```

```
7914                {'call': 'semtimedop',
7915                 'reason': set([('task_struct', 'cred'),
7916                                ('task_struct', 'io_context'),
7917                                ('task_struct', 'real_cred')])},
7918                {'call': 'umount',
7919                 'reason': set([('task_struct', 'cred'),
7920                                ('task_struct', 'io_context'),
7921                                ('task_struct', 'real_cred')])},
7922                {'call': 'sched_rr_get_interval',
7923                 'reason': set([('task_struct', 'cred'),
7924                                ('task_struct', 'io_context'),
7925                                ('task_struct', 'real_cred')])},
7926                {'call': 'rt_sigprocmask',
7927                 'reason': set([('task_struct', 'cred'),
7928                                ('task_struct', 'io_context'),
7929                                ('task_struct', 'real_cred')])},
7930                {'call': 'setsid',
7931                 'reason': set([('task_struct', 'cred'),
7932                                ('task_struct', 'io_context'),
7933                                ('task_struct', 'real_cred')])},
7934                {'call': 'sigaltstack',
7935                 'reason': set([('task_struct', 'cred'),
7936                                ('task_struct', 'io_context'),
7937                                ('task_struct', 'real_cred')])},
7938                {'call': 'sched_setattr',
7939                 'reason': set([('task_struct', 'cred'),
7940                                ('task_struct', 'io_context'),
7941                                ('task_struct', 'real_cred')])},
7942                {'call': 'migrate_pages',
7943                 'reason': set([('task_struct', 'cred'),
7944                                ('task_struct', 'io_context'),
7945                                ('task_struct', 'real_cred')])},
7946                {'call': 'getitimer',
7947                 'reason': set([('task_struct', 'cred'),
7948                                ('task_struct', 'io_context'),
7949                                ('task_struct', 'real_cred')])},
7950                {'call': 'setpgid',
7951                 'reason': set([('task_struct', 'cred'),
7952                                ('task_struct', 'io_context'),
7953                                ('task_struct', 'real_cred')])},
7954                {'call': 'getsid',
7955                 'reason': set([('task_struct', 'cred'),
7956                                ('task_struct', 'io_context'),
7957                                ('task_struct', 'real_cred')])},
7958                {'call': 'prlimit64',
7959                 'reason': set([('task_struct', 'cred'),
7960                                ('task_struct', 'io_context'),
7961                                ('task_struct', 'real_cred')])},
7962                {'call': 'perf_event_open',
7963                 'reason': set([('task_struct', 'cred'),
7964                                ('task_struct', 'io_context'),
7965                                ('task_struct', 'real_cred')])},
7966                {'call': 'rt_sigaction',
7967                 'reason': set([('task_struct', 'cred'),
7968                                ('task_struct', 'io_context'),
7969                                ('task_struct', 'real_cred')])},
7970                {'call': 'getpgid',
7971                 'reason': set([('task_struct', 'cred'),
7972                                ('task_struct', 'io_context'),
7973                                ('task_struct', 'real_cred')])},
7974                {'call': 'getpriority',
7975                 'reason': set([('task_struct', 'cred'),
7976                                ('task_struct', 'io_context'),
7977                                ('task_struct', 'real_cred')])},
7978                {'call': 'sigaction',
7979                 'reason': set([('task_struct', 'cred'),
```



```
7980                                ('task_struct', 'io_context'),
7981                                ('task_struct', 'real_cred')])},
7982                {'call': 'setns',
7983                 'reason': set([('task_struct', 'cred'),
7984                                ('task_struct', 'io_context'),
7985                                ('task_struct', 'real_cred')])},
7986                {'call': 'fork',
7987                 'reason': set([('task_struct', 'cred'),
7988                                ('task_struct', 'io_context'),
7989                                ('task_struct', 'real_cred')])},
7990                {'call': 'get_robust_list',
7991                 'reason': set([('task_struct', 'cred'),
7992                                ('task_struct', 'io_context'),
7993                                ('task_struct', 'real_cred')])},
7994                {'call': 'mq_timedsend',
7995                 'reason': set([('task_struct', 'cred'),
7996                                ('task_struct', 'io_context'),
7997                                ('task_struct', 'real_cred')])},
7998                {'call': 'sched_getscheduler',
7999                 'reason': set([('task_struct', 'cred'),
8000                                ('task_struct', 'io_context'),
8001                                ('task_struct', 'real_cred')])},
8002                {'call': 'ptrace',
8003                 'reason': set([('task_struct', 'cred'),
8004                                ('task_struct', 'io_context'),
8005                                ('task_struct', 'real_cred')])},
8006                {'call': 'sched_getattr',
8007                 'reason': set([('task_struct', 'cred'),
8008                                ('task_struct', 'io_context'),
8009                                ('task_struct', 'real_cred')])},
8010                {'call': 'getrusage',
8011                 'reason': set([('task_struct', 'cred'),
8012                                ('task_struct', 'io_context'),
8013                                ('task_struct', 'real_cred')])},
8014                {'call': 'sched_setscheduler',
8015                 'reason': set([('task_struct', 'cred'),
8016                                ('task_struct', 'io_context'),
8017                                ('task_struct', 'real_cred')])},
8018                {'call': 'setitimer',
8019                 'reason': set([('task_struct', 'cred'),
8020                                ('task_struct', 'io_context'),
8021                                ('task_struct', 'real_cred')])},
8022                {'call': 'ioprio_get',
8023                 'reason': set([('task_struct', 'cred'),
8024                                ('task_struct', 'io_context'),
8025                                ('task_struct', 'real_cred')])},
8026                {'call': 'vfork',
8027                 'reason': set([('task_struct', 'cred'),
8028                                ('task_struct', 'io_context'),
8029                                ('task_struct', 'real_cred')])},
8030                {'call': 'prctl',
8031                 'reason': set([('task_struct', 'cred'),
8032                                ('task_struct', 'io_context'),
8033                                ('task_struct', 'real_cred')])},
8034                {'call': 'move_pages',
8035                 'reason': set([('task_struct', 'cred'),
8036                                ('task_struct', 'io_context'),
8037                                ('task_struct', 'real_cred')])},
8038                {'call': 'setpriority',
8039                 'reason': set([('task_struct', 'cred'),
8040                                ('task_struct', 'io_context'),
8041                                ('task_struct', 'real_cred')])},
8042                {'call': 'clone',
8043                 'reason': set([('task_struct', 'cred'),
8044                                ('task_struct', 'io_context'),
8045                                ('task_struct', 'real_cred')])},
```

```
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'io_context'),
                                ('task_struct', 'real_cred')])}],
 'ioprio_set': [{'call': 'keyctl',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigtimedwait',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'msgrcv',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'kill',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getaffinity',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setparam',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ioprio_set',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'capget',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'signal',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'umount',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'cred'),
```



```
                                ('task_struct', 'real_cred')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred'),
```

```
8178                                ('task_struct', 'real_cred')])},
8179                {'call': 'setpriority',
8180                 'reason': set([('task_struct', 'cred'),
8181                                ('task_struct', 'real_cred')])},
8182                {'call': 'clone',
8183                 'reason': set([('task_struct', 'cred'),
8184                                ('task_struct', 'real_cred')])},
8185                {'call': 'sched_getparam',
8186                 'reason': set([('task_struct', 'cred'),
8187                                ('task_struct', 'real_cred')])}],
8188  'keyctl': [{'call': 'keyctl',
8189              'reason': set([('cred', 'session_keyring'),
8190                             ('key', 'description'),
8191                             ('key', 'gid'),
8192                             ('key', 'perm'),
8193                             ('key', 'quotalen'),
8194                             ('key', 'serial'),
8195                             ('key', 'uid'),
8196                             ('key_type', 'name'),
8197                             ('key_type', 'read'),
8198                             ('key_user', 'qnbytes'),
8199                             ('key_user', 'qnkeys'),
8200                             ('request_key_auth', 'target_key'),
8201                             ('task_struct', 'cred'),
8202                             ('task_struct', 'mm'),
8203                             ('task_struct', 'pid'),
8204                             ('task_struct', 'real_cred')])},
8205             {'call': 'rt_sigtimedwait',
8206              'reason': set([('task_struct', 'cred'),
8207                             ('task_struct', 'mm'),
8208                             ('task_struct', 'pid'),
8209                             ('task_struct', 'real_cred')])},
8210             {'call': 'setfsuid',
8211              'reason': set([('cred', 'session_keyring')])},
8212             {'call': 'msgrcv',
8213              'reason': set([('task_struct', 'cred'),
8214                             ('task_struct', 'mm'),
8215                             ('task_struct', 'pid'),
8216                             ('task_struct', 'real_cred')])},
8217             {'call': 'kill',
8218              'reason': set([('task_struct', 'cred'),
8219                             ('task_struct', 'mm'),
8220                             ('task_struct', 'pid'),
8221                             ('task_struct', 'real_cred')])},
8222             {'call': 'getresuid16',
8223              'reason': set([('cred', 'session_keyring')])},
8224             {'call': 'getresgid',
8225              'reason': set([('cred', 'session_keyring')])},
8226             {'call': 'sched_getaffinity',
8227              'reason': set([('task_struct', 'cred'),
8228                             ('task_struct', 'mm'),
8229                             ('task_struct', 'pid'),
8230                             ('task_struct', 'real_cred')])},
8231             {'call': 'sched_setparam',
8232              'reason': set([('task_struct', 'cred'),
8233                             ('task_struct', 'mm'),
8234                             ('task_struct', 'pid'),
8235                             ('task_struct', 'real_cred')])},
8236             {'call': 'setgid',
8237              'reason': set([('cred', 'egid'),
8238                             ('cred', 'gid'),
8239                             ('cred', 'session_keyring'),
8240                             ('cred', 'sgid')])},
8241             {'call': 'ioprio_set',
8242              'reason': set([('cred', 'session_keyring'),
8243                             ('task_struct', 'cred'),
```



```
8244                             ('task_struct', 'mm'),
8245                             ('task_struct', 'pid'),
8246                             ('task_struct', 'real_cred')])},
8247             {'call': 'capset', 'reason': set([('cred', 'session_keyring')])},
8248             {'call': 'getppid',
8249              'reason': set([('task_struct', 'cred'),
8250                             ('task_struct', 'mm'),
8251                             ('task_struct', 'pid'),
8252                             ('task_struct', 'real_cred')])},
8253             {'call': 'mq_timedreceive',
8254              'reason': set([('task_struct', 'cred'),
8255                             ('task_struct', 'mm'),
8256                             ('task_struct', 'pid'),
8257                             ('task_struct', 'real_cred')])},
8258             {'call': 'getresgid16',
8259              'reason': set([('cred', 'session_keyring')])},
8260             {'call': 'capget',
8261              'reason': set([('task_struct', 'cred'),
8262                             ('task_struct', 'mm'),
8263                             ('task_struct', 'pid'),
8264                             ('task_struct', 'real_cred')])},
8265             {'call': 'sched_setaffinity',
8266              'reason': set([('cred', 'session_keyring'),
8267                             ('task_struct', 'cred'),
8268                             ('task_struct', 'mm'),
8269                             ('task_struct', 'pid'),
8270                             ('task_struct', 'real_cred')])},
8271             {'call': 'setfsgid',
8272              'reason': set([('cred', 'session_keyring')])},
8273             {'call': 'unshare', 'reason': set([('cred', 'session_keyring')])},
8274             {'call': 'signal',
8275              'reason': set([('task_struct', 'cred'),
8276                             ('task_struct', 'mm'),
8277                             ('task_struct', 'pid'),
8278                             ('task_struct', 'real_cred')])},
8279             {'call': 'setreuid',
8280              'reason': set([('cred', 'euid'),
8281                             ('cred', 'session_keyring'),
8282                             ('cred', 'suid'),
8283                             ('cred', 'uid')])},
8284             {'call': 'semtimedop',
8285              'reason': set([('task_struct', 'cred'),
8286                             ('task_struct', 'mm'),
8287                             ('task_struct', 'pid'),
8288                             ('task_struct', 'real_cred')])},
8289             {'call': 'umount',
8290              'reason': set([('task_struct', 'cred'),
8291                             ('task_struct', 'mm'),
8292                             ('task_struct', 'pid'),
8293                             ('task_struct', 'real_cred')])},
8294             {'call': 'sched_rr_get_interval',
8295              'reason': set([('task_struct', 'cred'),
8296                             ('task_struct', 'mm'),
8297                             ('task_struct', 'pid'),
8298                             ('task_struct', 'real_cred')])},
8299             {'call': 'epoll_create1',
8300              'reason': set([('cred', 'session_keyring')])},
8301             {'call': 'getresuid',
8302              'reason': set([('cred', 'session_keyring')])},
8303             {'call': 'rt_sigprocmask',
8304              'reason': set([('task_struct', 'cred'),
8305                             ('task_struct', 'mm'),
8306                             ('task_struct', 'pid'),
8307                             ('task_struct', 'real_cred')])},
8308             {'call': 'setsid',
8309              'reason': set([('task_struct', 'cred'),
```

```
8310                            ('task_struct', 'mm'),
8311                            ('task_struct', 'pid'),
8312                            ('task_struct', 'real_cred')])},
8313            {'call': 'sigaltstack',
8314             'reason': set([('task_struct', 'cred'),
8315                            ('task_struct', 'mm'),
8316                            ('task_struct', 'pid'),
8317                            ('task_struct', 'real_cred')])},
8318            {'call': 'sched_setattr',
8319             'reason': set([('task_struct', 'cred'),
8320                            ('task_struct', 'mm'),
8321                            ('task_struct', 'pid'),
8322                            ('task_struct', 'real_cred')])},
8323            {'call': 'migrate_pages',
8324             'reason': set([('cred', 'session_keyring'),
8325                            ('task_struct', 'cred'),
8326                            ('task_struct', 'mm'),
8327                            ('task_struct', 'pid'),
8328                            ('task_struct', 'real_cred')])},
8329            {'call': 'getitimer',
8330             'reason': set([('task_struct', 'cred'),
8331                            ('task_struct', 'mm'),
8332                            ('task_struct', 'pid'),
8333                            ('task_struct', 'real_cred')])},
8334            {'call': 'setpgid',
8335             'reason': set([('task_struct', 'cred'),
8336                            ('task_struct', 'mm'),
8337                            ('task_struct', 'pid'),
8338                            ('task_struct', 'real_cred')])},
8339            {'call': 'setresgid',
8340             'reason': set([('cred', 'egid'),
8341                            ('cred', 'gid'),
8342                            ('cred', 'session_keyring'),
8343                            ('cred', 'sgid')])},
8344            {'call': 'setregid',
8345             'reason': set([('cred', 'egid'),
8346                            ('cred', 'gid'),
8347                            ('cred', 'session_keyring'),
8348                            ('cred', 'sgid')])},
8349            {'call': 'getsid',
8350             'reason': set([('task_struct', 'cred'),
8351                            ('task_struct', 'mm'),
8352                            ('task_struct', 'pid'),
8353                            ('task_struct', 'real_cred')])},
8354            {'call': 'prlimit64',
8355             'reason': set([('cred', 'session_keyring'),
8356                            ('task_struct', 'cred'),
8357                            ('task_struct', 'mm'),
8358                            ('task_struct', 'pid'),
8359                            ('task_struct', 'real_cred')])},
8360            {'call': 'perf_event_open',
8361             'reason': set([('task_struct', 'cred'),
8362                            ('task_struct', 'mm'),
8363                            ('task_struct', 'pid'),
8364                            ('task_struct', 'real_cred')])},
8365            {'call': 'getgroups16',
8366             'reason': set([('cred', 'session_keyring')])},
8367            {'call': 'rt_sigaction',
8368             'reason': set([('task_struct', 'cred'),
8369                            ('task_struct', 'mm'),
8370                            ('task_struct', 'pid'),
8371                            ('task_struct', 'real_cred')])},
8372            {'call': 'request_key',
8373             'reason': set([('key', 'description'),
8374                            ('key', 'perm'),
8375                            ('key', 'quotalen'),
```



```
8376                            ('key', 'serial'),
8377                            ('key_type', 'name'),
8378                            ('key_type', 'read')])},
8379            {'call': 'getpgid',
8380             'reason': set([('task_struct', 'cred'),
8381                            ('task_struct', 'mm'),
8382                            ('task_struct', 'pid'),
8383                            ('task_struct', 'real_cred')])},
8384            {'call': 'getpriority',
8385             'reason': set([('cred', 'session_keyring'),
8386                            ('task_struct', 'cred'),
8387                            ('task_struct', 'mm'),
8388                            ('task_struct', 'pid'),
8389                            ('task_struct', 'real_cred')])},
8390            {'call': 'sigaction',
8391             'reason': set([('task_struct', 'cred'),
8392                            ('task_struct', 'mm'),
8393                            ('task_struct', 'pid'),
8394                            ('task_struct', 'real_cred')])},
8395            {'call': 'faccessat',
8396             'reason': set([('cred', 'session_keyring')])},
8397            {'call': 'setns',
8398             'reason': set([('task_struct', 'cred'),
8399                            ('task_struct', 'mm'),
8400                            ('task_struct', 'pid'),
8401                            ('task_struct', 'real_cred')])},
8402            {'call': 'fork',
8403             'reason': set([('task_struct', 'cred'),
8404                            ('task_struct', 'mm'),
8405                            ('task_struct', 'pid'),
8406                            ('task_struct', 'real_cred')])},
8407            {'call': 'get_robust_list',
8408             'reason': set([('task_struct', 'cred'),
8409                            ('task_struct', 'mm'),
8410                            ('task_struct', 'pid'),
8411                            ('task_struct', 'real_cred')])},
8412            {'call': 'mq_timedsend',
8413             'reason': set([('task_struct', 'cred'),
8414                            ('task_struct', 'mm'),
8415                            ('task_struct', 'pid'),
8416                            ('task_struct', 'real_cred')])},
8417            {'call': 'sched_getscheduler',
8418             'reason': set([('task_struct', 'cred'),
8419                            ('task_struct', 'mm'),
8420                            ('task_struct', 'pid'),
8421                            ('task_struct', 'real_cred')])},
8422            {'call': 'ptrace',
8423             'reason': set([('task_struct', 'cred'),
8424                            ('task_struct', 'mm'),
8425                            ('task_struct', 'pid'),
8426                            ('task_struct', 'real_cred')])},
8427            {'call': 'sched_getattr',
8428             'reason': set([('task_struct', 'cred'),
8429                            ('task_struct', 'mm'),
8430                            ('task_struct', 'pid'),
8431                            ('task_struct', 'real_cred')])},
8432            {'call': 'getrusage',
8433             'reason': set([('task_struct', 'cred'),
8434                            ('task_struct', 'mm'),
8435                            ('task_struct', 'pid'),
8436                            ('task_struct', 'real_cred')])},
8437            {'call': 'sched_setscheduler',
8438             'reason': set([('task_struct', 'cred'),
8439                            ('task_struct', 'mm'),
8440                            ('task_struct', 'pid'),
8441                            ('task_struct', 'real_cred')])},
```

```
8442             {'call': 'setresuid',
8443              'reason': set([('cred', 'euid'),
8444                             ('cred', 'session_keyring'),
8445                             ('cred', 'suid'),
8446                             ('cred', 'uid')])},
8447             {'call': 'setitimer',
8448              'reason': set([('task_struct', 'cred'),
8449                             ('task_struct', 'mm'),
8450                             ('task_struct', 'pid'),
8451                             ('task_struct', 'real_cred')])},
8452             {'call': 'ioprio_get',
8453              'reason': set([('cred', 'session_keyring'),
8454                             ('task_struct', 'cred'),
8455                             ('task_struct', 'mm'),
8456                             ('task_struct', 'pid'),
8457                             ('task_struct', 'real_cred')])},
8458             {'call': 'vfork',
8459              'reason': set([('task_struct', 'cred'),
8460                             ('task_struct', 'mm'),
8461                             ('task_struct', 'pid'),
8462                             ('task_struct', 'real_cred')])},
8463             {'call': 'setuid',
8464              'reason': set([('cred', 'euid'),
8465                             ('cred', 'session_keyring'),
8466                             ('cred', 'suid'),
8467                             ('cred', 'uid')])},
8468             {'call': 'prctl',
8469              'reason': set([('task_struct', 'cred'),
8470                             ('task_struct', 'mm'),
8471                             ('task_struct', 'pid'),
8472                             ('task_struct', 'real_cred')])},
8473             {'call': 'move_pages',
8474              'reason': set([('task_struct', 'cred'),
8475                             ('task_struct', 'mm'),
8476                             ('task_struct', 'pid'),
8477                             ('task_struct', 'real_cred')])},
8478             {'call': 'getgroups',
8479              'reason': set([('cred', 'session_keyring')])},
8480             {'call': 'setpriority',
8481              'reason': set([('cred', 'session_keyring'),
8482                             ('task_struct', 'cred'),
8483                             ('task_struct', 'mm'),
8484                             ('task_struct', 'pid'),
8485                             ('task_struct', 'real_cred')])},
8486             {'call': 'clone',
8487              'reason': set([('task_struct', 'cred'),
8488                             ('task_struct', 'mm'),
8489                             ('task_struct', 'pid'),
8490                             ('task_struct', 'real_cred')])},
8491             {'call': 'sched_getparam',
8492              'reason': set([('task_struct', 'cred'),
8493                             ('task_struct', 'mm'),
8494                             ('task_struct', 'pid'),
8495                             ('task_struct', 'real_cred')])}],
8496  'kill': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
8497           {'call': 'rt_sigtimedwait',
8498            'reason': set([('task_struct', 'cred')])},
8499           {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
8500           {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
8501           {'call': 'sched_getaffinity',
8502            'reason': set([('task_struct', 'cred')])},
8503           {'call': 'sched_setparam',
8504            'reason': set([('task_struct', 'cred')])},
8505           {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
8506           {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
8507           {'call': 'mq_timedreceive',
```



```
8508            'reason': set([('task_struct', 'cred')])},
8509           {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
8510           {'call': 'sched_setaffinity',
8511            'reason': set([('task_struct', 'cred')])},
8512           {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
8513           {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
8514           {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
8515           {'call': 'sched_rr_get_interval',
8516            'reason': set([('task_struct', 'cred')])},
8517           {'call': 'rt_sigprocmask',
8518            'reason': set([('task_struct', 'cred')])},
8519           {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
8520           {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
8521           {'call': 'sched_setattr', 'reason': set([('task_struct', 'cred')])},
8522           {'call': 'migrate_pages', 'reason': set([('task_struct', 'cred')])},
8523           {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
8524           {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
8525           {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
8526           {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
8527           {'call': 'perf_event_open',
8528            'reason': set([('task_struct', 'cred')])},
8529           {'call': 'rt_sigaction', 'reason': set([('task_struct', 'cred')])},
8530           {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
8531           {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
8532           {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
8533           {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
8534           {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
8535           {'call': 'get_robust_list',
8536            'reason': set([('task_struct', 'cred')])},
8537           {'call': 'mq_timedsend', 'reason': set([('task_struct', 'cred')])},
8538           {'call': 'sched_getscheduler',
8539            'reason': set([('task_struct', 'cred')])},
8540           {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
8541           {'call': 'sched_getattr', 'reason': set([('task_struct', 'cred')])},
8542           {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
8543           {'call': 'sched_setscheduler',
8544            'reason': set([('task_struct', 'cred')])},
8545           {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
8546           {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
8547           {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
8548           {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
8549           {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
8550           {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
8551           {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
8552           {'call': 'sched_getparam',
8553            'reason': set([('task_struct', 'cred')])}],
8554  'lgetxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
8555                {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
8556                {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
8557                {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
8558                {'call': 'remap_file_pages',
8559                 'reason': set([('path', 'dentry')])},
8560                {'call': 'dup3', 'reason': set([('path', 'dentry')])},
8561                {'call': 'unshare', 'reason': set([('path', 'dentry')])},
8562                {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
8563                {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
8564                {'call': 'flock', 'reason': set([('path', 'dentry')])},
8565                {'call': 'openat', 'reason': set([('path', 'dentry')])},
8566                {'call': 'lookup_dcookie',
8567                 'reason': set([('path', 'dentry')])},
8568                {'call': 'uselib', 'reason': set([('path', 'dentry')])},
8569                {'call': 'accept4', 'reason': set([('path', 'dentry')])},
8570                {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
8571                {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
8572                {'call': 'shmat', 'reason': set([('path', 'dentry')])},
8573                {'call': 'socket', 'reason': set([('path', 'dentry')])},
```

```
8574                {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
8575                {'call': 'perf_event_open',
8576                 'reason': set([('path', 'dentry')])},
8577                {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
8578                {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
8579                {'call': 'acct', 'reason': set([('path', 'dentry')])},
8580                {'call': 'open', 'reason': set([('path', 'dentry')])},
8581                {'call': 'dup', 'reason': set([('path', 'dentry')])},
8582                {'call': 'setns', 'reason': set([('path', 'dentry')])},
8583                {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
8584                {'call': 'swapon', 'reason': set([('path', 'dentry')])},
8585                {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
8586                {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
8587                {'call': 'open_by_handle_at',
8588                 'reason': set([('path', 'dentry')])}],
8589  'linkat': [{'call': 'eventfd2', 'reason': set([('path', 'mnt')])},
8590             {'call': 'swapoff', 'reason': set([('path', 'mnt')])},
8591             {'call': 'pivot_root', 'reason': set([('path', 'mnt')])},
8592             {'call': 'memfd_create', 'reason': set([('path', 'mnt')])},
8593             {'call': 'remap_file_pages', 'reason': set([('path', 'mnt')])},
8594             {'call': 'dup3', 'reason': set([('path', 'mnt')])},
8595             {'call': 'unshare', 'reason': set([('path', 'mnt')])},
8596             {'call': 'epoll_create1', 'reason': set([('path', 'mnt')])},
8597             {'call': 'epoll_ctl', 'reason': set([('path', 'mnt')])},
8598             {'call': 'flock', 'reason': set([('path', 'mnt')])},
8599             {'call': 'openat', 'reason': set([('path', 'mnt')])},
8600             {'call': 'lookup_dcookie', 'reason': set([('path', 'mnt')])},
8601             {'call': 'uselib', 'reason': set([('path', 'mnt')])},
8602             {'call': 'accept4', 'reason': set([('path', 'mnt')])},
8603             {'call': 'socketpair', 'reason': set([('path', 'mnt')])},
8604             {'call': 'getcwd', 'reason': set([('path', 'mnt')])},
8605             {'call': 'shmat', 'reason': set([('path', 'mnt')])},
8606             {'call': 'socket', 'reason': set([('path', 'mnt')])},
8607             {'call': 'pipe2', 'reason': set([('path', 'mnt')])},
8608             {'call': 'perf_event_open', 'reason': set([('path', 'mnt')])},
8609             {'call': 'shmdt', 'reason': set([('path', 'mnt')])},
8610             {'call': 'quotactl', 'reason': set([('path', 'mnt')])},
8611             {'call': 'acct', 'reason': set([('path', 'mnt')])},
8612             {'call': 'open', 'reason': set([('path', 'mnt')])},
8613             {'call': 'dup', 'reason': set([('path', 'mnt')])},
8614             {'call': 'setns', 'reason': set([('path', 'mnt')])},
8615             {'call': 'shmctl', 'reason': set([('path', 'mnt')])},
8616             {'call': 'swapon', 'reason': set([('path', 'mnt')])},
8617             {'call': 'mmap_pgoff', 'reason': set([('path', 'mnt')])},
8618             {'call': 'mq_open', 'reason': set([('path', 'mnt')])},
8619             {'call': 'open_by_handle_at', 'reason': set([('path', 'mnt')])}],
8620  'listen': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
8621             {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
8622             {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
8623             {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
8624             {'call': 'readahead', 'reason': set([('fd', 'file')])},
8625             {'call': 'getdents', 'reason': set([('fd', 'file')])},
8626             {'call': 'writev', 'reason': set([('fd', 'file')])},
8627             {'call': 'preadv64', 'reason': set([('fd', 'file')])},
8628             {'call': 'fchmod', 'reason': set([('fd', 'file')])},
8629             {'call': 'pread64', 'reason': set([('fd', 'file')])},
8630             {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
8631             {'call': 'read', 'reason': set([('fd', 'file')])},
8632             {'call': 'fchown', 'reason': set([('fd', 'file')])},
8633             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
8634             {'call': 'utime', 'reason': set([('fd', 'file')])},
8635             {'call': 'fsync', 'reason': set([('fd', 'file')])},
8636             {'call': 'bpf', 'reason': set([('fd', 'file')])},
8637             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
8638             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
8639             {'call': 'sendto', 'reason': set([('fd', 'file')])},
```



```
8640             {'call': 'tee', 'reason': set([('fd', 'file')])},
8641             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
8642             {'call': 'lseek', 'reason': set([('fd', 'file')])},
8643             {'call': 'connect', 'reason': set([('fd', 'file')])},
8644             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
8645             {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
8646             {'call': 'flock', 'reason': set([('fd', 'file')])},
8647             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
8648             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
8649             {'call': 'accept4', 'reason': set([('fd', 'file')])},
8650             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
8651             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
8652             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
8653             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
8654             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
8655             {'call': 'splice', 'reason': set([('fd', 'file')])},
8656             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
8657             {'call': 'preadv', 'reason': set([('fd', 'file')])},
8658             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
8659             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
8660             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
8661             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
8662             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
8663             {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
8664             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
8665             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
8666             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
8667             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
8668             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
8669             {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
8670             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
8671             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
8672             {'call': 'listen', 'reason': set([('fd', 'file')])},
8673             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
8674             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
8675             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
8676             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
8677             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
8678             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
8679             {'call': 'llseek', 'reason': set([('fd', 'file')])},
8680             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
8681             {'call': 'readv', 'reason': set([('fd', 'file')])},
8682             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
8683             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
8684             {'call': 'write', 'reason': set([('fd', 'file')])},
8685             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
8686             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
8687             {'call': 'bind', 'reason': set([('fd', 'file')])},
8688             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
8689             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
8690  'listxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
8691                {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
8692                {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
8693                {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
8694                {'call': 'remap_file_pages',
8695                 'reason': set([('path', 'dentry')])},
8696                {'call': 'dup3', 'reason': set([('path', 'dentry')])},
8697                {'call': 'unshare', 'reason': set([('path', 'dentry')])},
8698                {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
8699                {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
8700                {'call': 'flock', 'reason': set([('path', 'dentry')])},
8701                {'call': 'openat', 'reason': set([('path', 'dentry')])},
8702                {'call': 'lookup_dcookie',
8703                 'reason': set([('path', 'dentry')])},
8704                {'call': 'uselib', 'reason': set([('path', 'dentry')])},
8705                {'call': 'accept4', 'reason': set([('path', 'dentry')])},
```

```
8706                {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
8707                {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
8708                {'call': 'shmat', 'reason': set([('path', 'dentry')])},
8709                {'call': 'socket', 'reason': set([('path', 'dentry')])},
8710                {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
8711                {'call': 'perf_event_open',
8712                 'reason': set([('path', 'dentry')])},
8713                {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
8714                {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
8715                {'call': 'acct', 'reason': set([('path', 'dentry')])},
8716                {'call': 'open', 'reason': set([('path', 'dentry')])},
8717                {'call': 'dup', 'reason': set([('path', 'dentry')])},
8718                {'call': 'setns', 'reason': set([('path', 'dentry')])},
8719                {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
8720                {'call': 'swapon', 'reason': set([('path', 'dentry')])},
8721                {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
8722                {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
8723                {'call': 'open_by_handle_at',
8724                 'reason': set([('path', 'dentry')])}],
8725  'llistxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
8726                 {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
8727                 {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
8728                 {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
8729                 {'call': 'remap_file_pages',
8730                  'reason': set([('path', 'dentry')])},
8731                 {'call': 'dup3', 'reason': set([('path', 'dentry')])},
8732                 {'call': 'unshare', 'reason': set([('path', 'dentry')])},
8733                 {'call': 'epoll_create1',
8734                  'reason': set([('path', 'dentry')])},
8735                 {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
8736                 {'call': 'flock', 'reason': set([('path', 'dentry')])},
8737                 {'call': 'openat', 'reason': set([('path', 'dentry')])},
8738                 {'call': 'lookup_dcookie',
8739                  'reason': set([('path', 'dentry')])},
8740                 {'call': 'uselib', 'reason': set([('path', 'dentry')])},
8741                 {'call': 'accept4', 'reason': set([('path', 'dentry')])},
8742                 {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
8743                 {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
8744                 {'call': 'shmat', 'reason': set([('path', 'dentry')])},
8745                 {'call': 'socket', 'reason': set([('path', 'dentry')])},
8746                 {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
8747                 {'call': 'perf_event_open',
8748                  'reason': set([('path', 'dentry')])},
8749                 {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
8750                 {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
8751                 {'call': 'acct', 'reason': set([('path', 'dentry')])},
8752                 {'call': 'open', 'reason': set([('path', 'dentry')])},
8753                 {'call': 'dup', 'reason': set([('path', 'dentry')])},
8754                 {'call': 'setns', 'reason': set([('path', 'dentry')])},
8755                 {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
8756                 {'call': 'swapon', 'reason': set([('path', 'dentry')])},
8757                 {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
8758                 {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
8759                 {'call': 'open_by_handle_at',
8760                  'reason': set([('path', 'dentry')])}],
8761  'llseek': [{'call': 'syncfs',
8762              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8763             {'call': 'vmsplice',
8764              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8765             {'call': 'pwritev64',
8766              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8767             {'call': 'fremovexattr',
8768              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8769             {'call': 'readahead',
8770              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8771             {'call': 'getdents',
```



```
8772              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8773             {'call': 'writev',
8774              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8775             {'call': 'preadv64',
8776              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8777             {'call': 'fchmod',
8778              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8779             {'call': 'pread64',
8780              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8781             {'call': 'signalfd4',
8782              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8783             {'call': 'read',
8784              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8785             {'call': 'fchown',
8786              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8787             {'call': 'mq_timedreceive',
8788              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8789             {'call': 'utime',
8790              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8791             {'call': 'fsync',
8792              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8793             {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
8794             {'call': 'recvfrom',
8795              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8796             {'call': 'fsetxattr',
8797              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8798             {'call': 'sendto',
8799              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8800             {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
8801             {'call': 'sync_file_range',
8802              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8803             {'call': 'lseek',
8804              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8805             {'call': 'connect',
8806              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8807             {'call': 'getsockname',
8808              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8809             {'call': 'epoll_ctl',
8810              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8811             {'call': 'flock',
8812              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8813             {'call': 'pwritev',
8814              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8815             {'call': 'fchdir',
8816              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8817             {'call': 'accept4',
8818              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8819             {'call': 'old_readdir',
8820              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8821             {'call': 'inotify_rm_watch',
8822              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8823             {'call': 'utimensat',
8824              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8825             {'call': 'inotify_add_watch',
8826              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8827             {'call': 'preadv2',
8828              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8829             {'call': 'splice',
8830              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8831             {'call': 'ftruncate',
8832              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8833             {'call': 'preadv',
8834              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8835             {'call': 'getpeername',
8836              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8837             {'call': 'setsockopt',
```

```
8838              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8839             {'call': 'fcntl',
8840              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8841             {'call': 'ioctl',
8842              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8843             {'call': 'pwrite64',
8844              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8845             {'call': 'perf_event_open',
8846              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8847             {'call': 'pwritev64v2',
8848              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8849             {'call': 'futimesat',
8850              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8851             {'call': 'pwritev2',
8852              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8853             {'call': 'shutdown',
8854              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8855             {'call': 'getsockopt',
8856              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8857             {'call': 'mq_getsetattr',
8858              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8859             {'call': 'fdatasync',
8860              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8861             {'call': 'getdents64',
8862              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8863             {'call': 'listen',
8864              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8865             {'call': 'copy_file_range',
8866              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8867             {'call': 'mq_timedsend',
8868              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8869             {'call': 'fgetxattr',
8870              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8871             {'call': 'fcntl64',
8872              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8873             {'call': 'fallocate',
8874              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8875             {'call': 'epoll_wait',
8876              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8877             {'call': 'llseek',
8878              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8879             {'call': 'preadv64v2',
8880              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8881             {'call': 'readv',
8882              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8883             {'call': 'fstatfs',
8884              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8885             {'call': 'fstatfs64',
8886              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8887             {'call': 'write',
8888              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8889             {'call': 'mq_notify',
8890              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8891             {'call': 'sendfile',
8892              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8893             {'call': 'bind',
8894              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8895             {'call': 'flistxattr',
8896              'reason': set([('fd', 'file'), ('fd', 'flags')])},
8897             {'call': 'sendfile64',
8898              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
8899  'lremovexattr': [{'call': 'eventfd2',
8900                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8901                   {'call': 'swapoff',
8902                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8903                   {'call': 'pivot_root',
```



```
8904                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8905                   {'call': 'memfd_create',
8906                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8907                   {'call': 'remap_file_pages',
8908                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8909                   {'call': 'dup3',
8910                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8911                   {'call': 'unshare',
8912                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8913                   {'call': 'epoll_create1',
8914                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8915                   {'call': 'epoll_ctl',
8916                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8917                   {'call': 'flock',
8918                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8919                   {'call': 'openat',
8920                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8921                   {'call': 'lookup_dcookie',
8922                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8923                   {'call': 'uselib',
8924                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8925                   {'call': 'accept4',
8926                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8927                   {'call': 'socketpair',
8928                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8929                   {'call': 'getcwd',
8930                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8931                   {'call': 'shmat',
8932                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8933                   {'call': 'socket',
8934                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8935                   {'call': 'pipe2',
8936                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8937                   {'call': 'perf_event_open',
8938                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8939                   {'call': 'shmdt',
8940                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8941                   {'call': 'quotactl',
8942                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8943                   {'call': 'acct',
8944                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8945                   {'call': 'open',
8946                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8947                   {'call': 'dup',
8948                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8949                   {'call': 'setns',
8950                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8951                   {'call': 'shmctl',
8952                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8953                   {'call': 'swapon',
8954                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8955                   {'call': 'mmap_pgoff',
8956                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8957                   {'call': 'mq_open',
8958                    'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8959                   {'call': 'open_by_handle_at',
8960                    'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
8961  'lseek': [{'call': 'syncfs',
8962             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8963            {'call': 'vmsplice',
8964             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8965            {'call': 'pwritev64',
8966             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8967            {'call': 'fremovexattr',
8968             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8969            {'call': 'readahead',
```

```
8970             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8971            {'call': 'getdents',
8972             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8973            {'call': 'writev',
8974             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8975            {'call': 'preadv64',
8976             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8977            {'call': 'fchmod',
8978             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8979            {'call': 'pread64',
8980             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8981            {'call': 'signalfd4',
8982             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8983            {'call': 'read', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
8984            {'call': 'fchown',
8985             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8986            {'call': 'mq_timedreceive',
8987             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8988            {'call': 'utime',
8989             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8990            {'call': 'fsync',
8991             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8992            {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
8993            {'call': 'recvfrom',
8994             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8995            {'call': 'fsetxattr',
8996             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8997            {'call': 'sendto',
8998             'reason': set([('fd', 'file'), ('fd', 'flags')])},
8999            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
9000            {'call': 'sync_file_range',
9001             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9002            {'call': 'lseek',
9003             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9004            {'call': 'connect',
9005             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9006            {'call': 'getsockname',
9007             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9008            {'call': 'epoll_ctl',
9009             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9010            {'call': 'flock',
9011             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9012            {'call': 'pwritev',
9013             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9014            {'call': 'fchdir',
9015             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9016            {'call': 'accept4',
9017             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9018            {'call': 'old_readdir',
9019             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9020            {'call': 'inotify_rm_watch',
9021             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9022            {'call': 'utimensat',
9023             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9024            {'call': 'inotify_add_watch',
9025             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9026            {'call': 'preadv2',
9027             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9028            {'call': 'splice',
9029             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9030            {'call': 'ftruncate',
9031             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9032            {'call': 'preadv',
9033             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9034            {'call': 'getpeername',
9035             'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
9036            {'call': 'setsockopt',
9037             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9038            {'call': 'fcntl',
9039             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9040            {'call': 'ioctl',
9041             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9042            {'call': 'pwrite64',
9043             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9044            {'call': 'perf_event_open',
9045             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9046            {'call': 'pwritev64v2',
9047             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9048            {'call': 'futimesat',
9049             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9050            {'call': 'pwritev2',
9051             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9052            {'call': 'shutdown',
9053             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9054            {'call': 'getsockopt',
9055             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9056            {'call': 'mq_getsetattr',
9057             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9058            {'call': 'fdatasync',
9059             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9060            {'call': 'getdents64',
9061             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9062            {'call': 'listen',
9063             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9064            {'call': 'copy_file_range',
9065             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9066            {'call': 'mq_timedsend',
9067             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9068            {'call': 'fgetxattr',
9069             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9070            {'call': 'fcntl64',
9071             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9072            {'call': 'fallocate',
9073             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9074            {'call': 'epoll_wait',
9075             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9076            {'call': 'llseek',
9077             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9078            {'call': 'preadv64v2',
9079             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9080            {'call': 'readv',
9081             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9082            {'call': 'fstatfs',
9083             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9084            {'call': 'fstatfs64',
9085             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9086            {'call': 'write',
9087             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9088            {'call': 'mq_notify',
9089             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9090            {'call': 'sendfile',
9091             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9092            {'call': 'bind', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
9093            {'call': 'flistxattr',
9094             'reason': set([('fd', 'file'), ('fd', 'flags')])},
9095            {'call': 'sendfile64',
9096             'reason': set([('fd', 'file'), ('fd', 'flags')])}],
9097  'lsetxattr': [{'call': 'eventfd2',
9098                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9099                {'call': 'swapoff',
9100                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9101                {'call': 'pivot_root',
```

```
9102                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9103                {'call': 'memfd_create',
9104                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9105                {'call': 'remap_file_pages',
9106                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9107                {'call': 'dup3',
9108                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9109                {'call': 'unshare',
9110                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9111                {'call': 'epoll_create1',
9112                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9113                {'call': 'epoll_ctl',
9114                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9115                {'call': 'flock',
9116                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9117                {'call': 'openat',
9118                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9119                {'call': 'lookup_dcookie',
9120                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9121                {'call': 'uselib',
9122                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9123                {'call': 'accept4',
9124                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9125                {'call': 'socketpair',
9126                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9127                {'call': 'getcwd',
9128                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9129                {'call': 'shmat',
9130                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9131                {'call': 'socket',
9132                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9133                {'call': 'pipe2',
9134                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9135                {'call': 'perf_event_open',
9136                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9137                {'call': 'shmdt',
9138                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9139                {'call': 'quotactl',
9140                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9141                {'call': 'acct',
9142                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9143                {'call': 'open',
9144                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9145                {'call': 'dup',
9146                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9147                {'call': 'setns',
9148                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9149                {'call': 'shmctl',
9150                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9151                {'call': 'swapon',
9152                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9153                {'call': 'mmap_pgoff',
9154                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9155                {'call': 'mq_open',
9156                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9157                {'call': 'open_by_handle_at',
9158                 'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
9159  'lstat': [{'call': 'lstat',
9160             'reason': set([('__old_kernel_stat', 'st_ino'),
9161                            ('__old_kernel_stat', 'st_nlink')])},
9162            {'call': 'stat',
9163             'reason': set([('__old_kernel_stat', 'st_ino'),
9164                            ('__old_kernel_stat', 'st_nlink')])},
9165            {'call': 'fstat',
9166             'reason': set([('__old_kernel_stat', 'st_ino'),
9167                            ('__old_kernel_stat', 'st_nlink'),
```



```
9168                            ('kstat', 'dev'),
9169                            ('kstat', 'ino'),
9170                            ('kstat', 'nlink'),
9171                            ('kstat', 'rdev')])},
9172           {'call': 'newfstat',
9173            'reason': set([('kstat', 'dev'),
9174                           ('kstat', 'ino'),
9175                           ('kstat', 'nlink'),
9176                           ('kstat', 'rdev')])}],
9177  'madvise': [{'call': 'remap_file_pages',
9178               'reason': set([('vm_area_struct', 'vm_end'),
9179                              ('vm_area_struct', 'vm_start')])},
9180              {'call': 'shmdt',
9181               'reason': set([('vm_area_struct', 'vm_end'),
9182                              ('vm_area_struct', 'vm_start')])},
9183              {'call': 'brk',
9184               'reason': set([('vm_area_struct', 'vm_end'),
9185                              ('vm_area_struct', 'vm_start')])},
9186              {'call': 'get_mempolicy',
9187               'reason': set([('vm_area_struct', 'vm_end'),
9188                              ('vm_area_struct', 'vm_start')])},
9189              {'call': 'munlockall',
9190               'reason': set([('vm_area_struct', 'vm_end'),
9191                              ('vm_area_struct', 'vm_start')])},
9192              {'call': 'pkey_mprotect',
9193               'reason': set([('vm_area_struct', 'vm_end'),
9194                              ('vm_area_struct', 'vm_start')])},
9195              {'call': 'madvise',
9196               'reason': set([('vm_area_struct', 'vm_end'),
9197                              ('vm_area_struct', 'vm_start')])},
9198              {'call': 'mprotect',
9199               'reason': set([('vm_area_struct', 'vm_end'),
9200                              ('vm_area_struct', 'vm_start')])},
9201              {'call': 'mremap',
9202               'reason': set([('vm_area_struct', 'vm_end'),
9203                              ('vm_area_struct', 'vm_start')])},
9204              {'call': 'prctl',
9205               'reason': set([('vm_area_struct', 'vm_end'),
9206                              ('vm_area_struct', 'vm_start')])},
9207              {'call': 'munlock',
9208               'reason': set([('vm_area_struct', 'vm_end'),
9209                              ('vm_area_struct', 'vm_start')])},
9210              {'call': 'mincore',
9211               'reason': set([('vm_area_struct', 'vm_end'),
9212                              ('vm_area_struct', 'vm_start')])},
9213              {'call': 'mlockall',
9214               'reason': set([('vm_area_struct', 'vm_end'),
9215                              ('vm_area_struct', 'vm_start')])}],
9216  'migrate_pages': [{'call': 'keyctl',
9217                     'reason': set([('mm_segment_t', 'seg'),
9218                                    ('task_struct', 'cred'),
9219                                    ('task_struct', 'real_cred')])},
9220                    {'call': 'rt_sigtimedwait',
9221                     'reason': set([('mm_segment_t', 'seg'),
9222                                    ('task_struct', 'cred'),
9223                                    ('task_struct', 'real_cred')])},
9224                    {'call': 'msgrcv',
9225                     'reason': set([('mm_segment_t', 'seg'),
9226                                    ('task_struct', 'cred'),
9227                                    ('task_struct', 'real_cred')])},
9228                    {'call': 'kill',
9229                     'reason': set([('mm_segment_t', 'seg'),
9230                                    ('task_struct', 'cred'),
9231                                    ('task_struct', 'real_cred')])},
9232                    {'call': 'sched_getaffinity',
9233                     'reason': set([('mm_segment_t', 'seg'),
```

```
9234                                    ('task_struct', 'cred'),
9235                                    ('task_struct', 'real_cred')])},
9236                    {'call': 'sched_setparam',
9237                     'reason': set([('mm_segment_t', 'seg'),
9238                                    ('task_struct', 'cred'),
9239                                    ('task_struct', 'real_cred')])},
9240                    {'call': 'ioprio_set',
9241                     'reason': set([('mm_segment_t', 'seg'),
9242                                    ('task_struct', 'cred'),
9243                                    ('task_struct', 'real_cred')])},
9244                    {'call': 'getppid',
9245                     'reason': set([('mm_segment_t', 'seg'),
9246                                    ('task_struct', 'cred'),
9247                                    ('task_struct', 'real_cred')])},
9248                    {'call': 'ioperm',
9249                     'reason': set([('mm_segment_t', 'seg')])},
9250                    {'call': 'mq_timedreceive',
9251                     'reason': set([('mm_segment_t', 'seg'),
9252                                    ('task_struct', 'cred'),
9253                                    ('task_struct', 'real_cred')])},
9254                    {'call': 'capget',
9255                     'reason': set([('mm_segment_t', 'seg'),
9256                                    ('task_struct', 'cred'),
9257                                    ('task_struct', 'real_cred')])},
9258                    {'call': 'sched_setaffinity',
9259                     'reason': set([('mm_segment_t', 'seg'),
9260                                    ('task_struct', 'cred'),
9261                                    ('task_struct', 'real_cred')])},
9262                    {'call': 'signal',
9263                     'reason': set([('mm_segment_t', 'seg'),
9264                                    ('task_struct', 'cred'),
9265                                    ('task_struct', 'real_cred')])},
9266                    {'call': 'setreuid',
9267                     'reason': set([('cred', 'euid'),
9268                                    ('cred', 'suid'),
9269                                    ('cred', 'uid')])},
9270                    {'call': 'semtimedop',
9271                     'reason': set([('mm_segment_t', 'seg'),
9272                                    ('task_struct', 'cred'),
9273                                    ('task_struct', 'real_cred')])},
9274                    {'call': 'umount',
9275                     'reason': set([('mm_segment_t', 'seg'),
9276                                    ('task_struct', 'cred'),
9277                                    ('task_struct', 'real_cred')])},
9278                    {'call': 'sched_rr_get_interval',
9279                     'reason': set([('mm_segment_t', 'seg'),
9280                                    ('task_struct', 'cred'),
9281                                    ('task_struct', 'real_cred')])},
9282                    {'call': 'rt_sigprocmask',
9283                     'reason': set([('mm_segment_t', 'seg'),
9284                                    ('task_struct', 'cred'),
9285                                    ('task_struct', 'real_cred')])},
9286                    {'call': 'setsid',
9287                     'reason': set([('mm_segment_t', 'seg'),
9288                                    ('task_struct', 'cred'),
9289                                    ('task_struct', 'real_cred')])},
9290                    {'call': 'sigaltstack',
9291                     'reason': set([('mm_segment_t', 'seg'),
9292                                    ('task_struct', 'cred'),
9293                                    ('task_struct', 'real_cred')])},
9294                    {'call': 'sched_setattr',
9295                     'reason': set([('mm_segment_t', 'seg'),
9296                                    ('task_struct', 'cred'),
9297                                    ('task_struct', 'real_cred')])},
9298                    {'call': 'migrate_pages',
9299                     'reason': set([('mm_segment_t', 'seg'),
```



```
9300                                    ('task_struct', 'cred'),
9301                                    ('task_struct', 'real_cred')])},
9302                    {'call': 'getitimer',
9303                     'reason': set([('mm_segment_t', 'seg'),
9304                                    ('task_struct', 'cred'),
9305                                    ('task_struct', 'real_cred')])},
9306                    {'call': 'setpgid',
9307                     'reason': set([('mm_segment_t', 'seg'),
9308                                    ('task_struct', 'cred'),
9309                                    ('task_struct', 'real_cred')])},
9310                    {'call': 'getsid',
9311                     'reason': set([('mm_segment_t', 'seg'),
9312                                    ('task_struct', 'cred'),
9313                                    ('task_struct', 'real_cred')])},
9314                    {'call': 'prlimit64',
9315                     'reason': set([('mm_segment_t', 'seg'),
9316                                    ('task_struct', 'cred'),
9317                                    ('task_struct', 'real_cred')])},
9318                    {'call': 'perf_event_open',
9319                     'reason': set([('mm_segment_t', 'seg'),
9320                                    ('task_struct', 'cred'),
9321                                    ('task_struct', 'real_cred')])},
9322                    {'call': 'rt_sigaction',
9323                     'reason': set([('mm_segment_t', 'seg'),
9324                                    ('task_struct', 'cred'),
9325                                    ('task_struct', 'real_cred')])},
9326                    {'call': 'getpgid',
9327                     'reason': set([('mm_segment_t', 'seg'),
9328                                    ('task_struct', 'cred'),
9329                                    ('task_struct', 'real_cred')])},
9330                    {'call': 'getpriority',
9331                     'reason': set([('mm_segment_t', 'seg'),
9332                                    ('task_struct', 'cred'),
9333                                    ('task_struct', 'real_cred')])},
9334                    {'call': 'sigaction',
9335                     'reason': set([('mm_segment_t', 'seg'),
9336                                    ('task_struct', 'cred'),
9337                                    ('task_struct', 'real_cred')])},
9338                    {'call': 'setns',
9339                     'reason': set([('mm_segment_t', 'seg'),
9340                                    ('task_struct', 'cred'),
9341                                    ('task_struct', 'real_cred')])},
9342                    {'call': 'fork',
9343                     'reason': set([('mm_segment_t', 'seg'),
9344                                    ('task_struct', 'cred'),
9345                                    ('task_struct', 'real_cred')])},
9346                    {'call': 'get_robust_list',
9347                     'reason': set([('mm_segment_t', 'seg'),
9348                                    ('task_struct', 'cred'),
9349                                    ('task_struct', 'real_cred')])},
9350                    {'call': 'mq_timedsend',
9351                     'reason': set([('mm_segment_t', 'seg'),
9352                                    ('task_struct', 'cred'),
9353                                    ('task_struct', 'real_cred')])},
9354                    {'call': 'sched_getscheduler',
9355                     'reason': set([('mm_segment_t', 'seg'),
9356                                    ('task_struct', 'cred'),
9357                                    ('task_struct', 'real_cred')])},
9358                    {'call': 'ptrace',
9359                     'reason': set([('mm_segment_t', 'seg'),
9360                                    ('task_struct', 'cred'),
9361                                    ('task_struct', 'real_cred')])},
9362                    {'call': 'sched_getattr',
9363                     'reason': set([('mm_segment_t', 'seg'),
9364                                    ('task_struct', 'cred'),
9365                                    ('task_struct', 'real_cred')])},
```

```
                    {'call': 'getrusage',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'setresuid',
                     'reason': set([('cred', 'euid'),
                                    ('cred', 'suid'),
                                    ('cred', 'uid')])},
                    {'call': 'setitimer',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'ioprio_get',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'vfork',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'setuid',
                     'reason': set([('cred', 'euid'),
                                    ('cred', 'suid'),
                                    ('cred', 'uid')])},
                    {'call': 'prctl',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'move_pages',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'setpriority',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'clone',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'sched_getparam',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])}],
 'mincore': [{'call': 'keyctl',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'rt_sigtimedwait',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'msgrcv',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'kill',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_getaffinity',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_setparam',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
```



```
             {'call': 'ioprio_set',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'remap_file_pages',
              'reason': set([('vm_area_struct', 'vm_start')])},
             {'call': 'getppid',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'mq_timedreceive',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'capget',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_setaffinity',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'signal',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'semtimedop',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'umount',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_rr_get_interval',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'rt_sigprocmask',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'setsid',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sigaltstack',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_setattr',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'migrate_pages',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'getitimer',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'setpgid',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'getsid',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'prlimit64',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'perf_event_open',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'shmdt',
              'reason': set([('vm_area_struct', 'vm_start')])},
             {'call': 'rt_sigaction',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'getpgid',
```

```
9498                'reason': set([('mm_segment_t', 'seg'),
9499                               ('task_struct', 'mm')])},
9500            {'call': 'brk', 'reason': set([('vm_area_struct', 'vm_start')])},
9501            {'call': 'getpriority',
9502             'reason': set([('mm_segment_t', 'seg'),
9503                            ('task_struct', 'mm')])},
9504            {'call': 'sigaction',
9505             'reason': set([('mm_segment_t', 'seg'),
9506                            ('task_struct', 'mm')])},
9507            {'call': 'setns',
9508             'reason': set([('mm_segment_t', 'seg'),
9509                            ('task_struct', 'mm')])},
9510            {'call': 'fork',
9511             'reason': set([('mm_segment_t', 'seg'),
9512                            ('task_struct', 'mm')])},
9513            {'call': 'get_mempolicy',
9514             'reason': set([('vm_area_struct', 'vm_start')])},
9515            {'call': 'get_robust_list',
9516             'reason': set([('mm_segment_t', 'seg'),
9517                            ('task_struct', 'mm')])},
9518            {'call': 'mq_timedsend',
9519             'reason': set([('mm_segment_t', 'seg'),
9520                            ('task_struct', 'mm')])},
9521            {'call': 'sched_getscheduler',
9522             'reason': set([('mm_segment_t', 'seg'),
9523                            ('task_struct', 'mm')])},
9524            {'call': 'ptrace',
9525             'reason': set([('mm_segment_t', 'seg'),
9526                            ('task_struct', 'mm')])},
9527            {'call': 'munlockall',
9528             'reason': set([('vm_area_struct', 'vm_start')])},
9529            {'call': 'pkey_mprotect',
9530             'reason': set([('vm_area_struct', 'vm_start')])},
9531            {'call': 'madvise',
9532             'reason': set([('vm_area_struct', 'vm_start')])},
9533            {'call': 'sched_getattr',
9534             'reason': set([('mm_segment_t', 'seg'),
9535                            ('task_struct', 'mm')])},
9536            {'call': 'getrusage',
9537             'reason': set([('mm_segment_t', 'seg'),
9538                            ('task_struct', 'mm')])},
9539            {'call': 'sched_setscheduler',
9540             'reason': set([('mm_segment_t', 'seg'),
9541                            ('task_struct', 'mm')])},
9542            {'call': 'setitimer',
9543             'reason': set([('mm_segment_t', 'seg'),
9544                            ('task_struct', 'mm')])},
9545            {'call': 'ioprio_get',
9546             'reason': set([('mm_segment_t', 'seg'),
9547                            ('task_struct', 'mm')])},
9548            {'call': 'vfork',
9549             'reason': set([('mm_segment_t', 'seg'),
9550                            ('task_struct', 'mm')])},
9551            {'call': 'mprotect',
9552             'reason': set([('vm_area_struct', 'vm_start')])},
9553            {'call': 'mremap',
9554             'reason': set([('vm_area_struct', 'vm_start')])},
9555            {'call': 'prctl',
9556             'reason': set([('mm_segment_t', 'seg'),
9557                            ('task_struct', 'mm'),
9558                            ('vm_area_struct', 'vm_start')])},
9559            {'call': 'move_pages',
9560             'reason': set([('mm_segment_t', 'seg'),
9561                            ('task_struct', 'mm')])},
9562            {'call': 'munlock',
9563             'reason': set([('vm_area_struct', 'vm_start')])},
```



```
9564            {'call': 'setpriority',
9565             'reason': set([('mm_segment_t', 'seg'),
9566                            ('task_struct', 'mm')])},
9567            {'call': 'mincore',
9568             'reason': set([('vm_area_struct', 'vm_start')])},
9569            {'call': 'clone',
9570             'reason': set([('mm_segment_t', 'seg'),
9571                            ('task_struct', 'mm')])},
9572            {'call': 'sched_getparam',
9573             'reason': set([('mm_segment_t', 'seg'),
9574                            ('task_struct', 'mm')])},
9575            {'call': 'mlockall',
9576             'reason': set([('vm_area_struct', 'vm_start')])}],
9577 'mkdirat': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
9578             {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
9579             {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
9580             {'call': 'quotactl',
9581              'reason': set([('super_block', 's_flags')])},
9582             {'call': 'swapon', 'reason': set([('super_block', 's_flags')])}],
9583 'mknodat': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
9584             {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
9585             {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
9586             {'call': 'quotactl',
9587              'reason': set([('super_block', 's_flags')])},
9588             {'call': 'swapon', 'reason': set([('super_block', 's_flags')])}],
9589 'mlock': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
9590           {'call': 'rt_sigtimedwait',
9591            'reason': set([('task_struct', 'mm')])},
9592           {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
9593           {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
9594           {'call': 'sched_getaffinity',
9595            'reason': set([('task_struct', 'mm')])},
9596           {'call': 'sched_setparam', 'reason': set([('task_struct', 'mm')])},
9597           {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
9598           {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
9599           {'call': 'mq_timedreceive',
9600            'reason': set([('task_struct', 'mm')])},
9601           {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
9602           {'call': 'sched_setaffinity',
9603            'reason': set([('task_struct', 'mm')])},
9604           {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
9605           {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
9606           {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
9607           {'call': 'sched_rr_get_interval',
9608            'reason': set([('task_struct', 'mm')])},
9609           {'call': 'rt_sigprocmask', 'reason': set([('task_struct', 'mm')])},
9610           {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
9611           {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
9612           {'call': 'sched_setattr', 'reason': set([('task_struct', 'mm')])},
9613           {'call': 'migrate_pages', 'reason': set([('task_struct', 'mm')])},
9614           {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
9615           {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
9616           {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
9617           {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
9618           {'call': 'perf_event_open',
9619            'reason': set([('task_struct', 'mm')])},
9620           {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
9621           {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
9622           {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
9623           {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
9624           {'call': 'setns', 'reason': set([('task_struct', 'mm')])},
9625           {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
9626           {'call': 'get_robust_list',
9627            'reason': set([('task_struct', 'mm')])},
9628           {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
9629           {'call': 'sched_getscheduler',
```

```
9630             'reason': set([('task_struct', 'mm')])},
9631            {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
9632            {'call': 'sched_getattr', 'reason': set([('task_struct', 'mm')])},
9633            {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
9634            {'call': 'sched_setscheduler',
9635             'reason': set([('task_struct', 'mm')])},
9636            {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
9637            {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
9638            {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
9639            {'call': 'prctl', 'reason': set([('task_struct', 'mm')])},
9640            {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
9641            {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
9642            {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
9643            {'call': 'sched_getparam', 'reason': set([('task_struct', 'mm')])}],
9644  'mlock2': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
9645             {'call': 'rt_sigtimedwait',
9646              'reason': set([('task_struct', 'mm')])},
9647             {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
9648             {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
9649             {'call': 'sched_getaffinity',
9650              'reason': set([('task_struct', 'mm')])},
9651             {'call': 'sched_setparam',
9652              'reason': set([('task_struct', 'mm')])},
9653             {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
9654             {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
9655             {'call': 'mq_timedreceive',
9656              'reason': set([('task_struct', 'mm')])},
9657             {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
9658             {'call': 'sched_setaffinity',
9659              'reason': set([('task_struct', 'mm')])},
9660             {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
9661             {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
9662             {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
9663             {'call': 'sched_rr_get_interval',
9664              'reason': set([('task_struct', 'mm')])},
9665             {'call': 'rt_sigprocmask',
9666              'reason': set([('task_struct', 'mm')])},
9667             {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
9668             {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
9669             {'call': 'sched_setattr', 'reason': set([('task_struct', 'mm')])},
9670             {'call': 'migrate_pages', 'reason': set([('task_struct', 'mm')])},
9671             {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
9672             {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
9673             {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
9674             {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
9675             {'call': 'perf_event_open',
9676              'reason': set([('task_struct', 'mm')])},
9677             {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
9678             {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
9679             {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
9680             {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
9681             {'call': 'setns', 'reason': set([('task_struct', 'mm')])},
9682             {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
9683             {'call': 'get_robust_list',
9684              'reason': set([('task_struct', 'mm')])},
9685             {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
9686             {'call': 'sched_getscheduler',
9687              'reason': set([('task_struct', 'mm')])},
9688             {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
9689             {'call': 'sched_getattr', 'reason': set([('task_struct', 'mm')])},
9690             {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
9691             {'call': 'sched_setscheduler',
9692              'reason': set([('task_struct', 'mm')])},
9693             {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
9694             {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
9695             {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
```



```
9696            {'call': 'prctl', 'reason': set([('task_struct', 'mm')])},
9697            {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
9698            {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
9699            {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
9700            {'call': 'sched_getparam',
9701             'reason': set([('task_struct', 'mm')])}],
9702  'mlockall': [{'call': 'keyctl',
9703                'reason': set([('task_struct', 'personality')])},
9704               {'call': 'rt_sigtimedwait',
9705                'reason': set([('task_struct', 'personality')])},
9706               {'call': 'msgrcv',
9707                'reason': set([('task_struct', 'personality')])},
9708               {'call': 'kill',
9709                'reason': set([('task_struct', 'personality')])},
9710               {'call': 'swapoff', 'reason': set([('mm_struct', 'total_vm')])},
9711               {'call': 'sched_getaffinity',
9712                'reason': set([('task_struct', 'personality')])},
9713               {'call': 'sched_setparam',
9714                'reason': set([('task_struct', 'personality')])},
9715               {'call': 'ioprio_set',
9716                'reason': set([('task_struct', 'personality')])},
9717               {'call': 'personality',
9718                'reason': set([('task_struct', 'personality')])},
9719               {'call': 'remap_file_pages',
9720                'reason': set([('mm_struct', 'total_vm'),
9721                               ('vm_area_struct', 'vm_end'),
9722                               ('vm_area_struct', 'vm_start')])},
9723               {'call': 'io_getevents',
9724                'reason': set([('mm_struct', 'total_vm')])},
9725               {'call': 'getppid',
9726                'reason': set([('task_struct', 'personality')])},
9727               {'call': 'mq_timedreceive',
9728                'reason': set([('task_struct', 'personality')])},
9729               {'call': 'capget',
9730                'reason': set([('task_struct', 'personality')])},
9731               {'call': 'sched_setaffinity',
9732                'reason': set([('task_struct', 'personality')])},
9733               {'call': 'signal',
9734                'reason': set([('task_struct', 'personality')])},
9735               {'call': 'semtimedop',
9736                'reason': set([('task_struct', 'personality')])},
9737               {'call': 'umount',
9738                'reason': set([('task_struct', 'personality')])},
9739               {'call': 'sched_rr_get_interval',
9740                'reason': set([('task_struct', 'personality')])},
9741               {'call': 'rt_sigprocmask',
9742                'reason': set([('task_struct', 'personality')])},
9743               {'call': 'setsid',
9744                'reason': set([('task_struct', 'personality')])},
9745               {'call': 'sigaltstack',
9746                'reason': set([('task_struct', 'personality')])},
9747               {'call': 'sched_setattr',
9748                'reason': set([('task_struct', 'personality')])},
9749               {'call': 'migrate_pages',
9750                'reason': set([('mm_struct', 'total_vm'),
9751                               ('task_struct', 'personality')])},
9752               {'call': 'getitimer',
9753                'reason': set([('task_struct', 'personality')])},
9754               {'call': 'setpgid',
9755                'reason': set([('task_struct', 'personality')])},
9756               {'call': 'getsid',
9757                'reason': set([('task_struct', 'personality')])},
9758               {'call': 'prlimit64',
9759                'reason': set([('task_struct', 'personality')])},
9760               {'call': 'perf_event_open',
9761                'reason': set([('task_struct', 'personality')])},
```

```
9762              {'call': 'shmdt',
9763               'reason': set([('mm_struct', 'total_vm'),
9764                              ('vm_area_struct', 'vm_end'),
9765                              ('vm_area_struct', 'vm_start')])},
9766              {'call': 'rt_sigaction',
9767               'reason': set([('task_struct', 'personality')])},
9768              {'call': 'getpgid',
9769               'reason': set([('task_struct', 'personality')])},
9770              {'call': 'brk',
9771               'reason': set([('mm_struct', 'total_vm'),
9772                              ('vm_area_struct', 'vm_end'),
9773                              ('vm_area_struct', 'vm_start')])},
9774              {'call': 'getpriority',
9775               'reason': set([('task_struct', 'personality')])},
9776              {'call': 'sigaction',
9777               'reason': set([('task_struct', 'personality')])},
9778              {'call': 'setns',
9779               'reason': set([('task_struct', 'personality')])},
9780              {'call': 'fork',
9781               'reason': set([('task_struct', 'personality')])},
9782              {'call': 'get_mempolicy',
9783               'reason': set([('mm_struct', 'total_vm'),
9784                              ('vm_area_struct', 'vm_end'),
9785                              ('vm_area_struct', 'vm_start')])},
9786              {'call': 'get_robust_list',
9787               'reason': set([('task_struct', 'personality')])},
9788              {'call': 'mq_timedsend',
9789               'reason': set([('task_struct', 'personality')])},
9790              {'call': 'sched_getscheduler',
9791               'reason': set([('task_struct', 'personality')])},
9792              {'call': 'ptrace',
9793               'reason': set([('task_struct', 'personality')])},
9794              {'call': 'munlockall',
9795               'reason': set([('vm_area_struct', 'vm_end'),
9796                              ('vm_area_struct', 'vm_start')])},
9797              {'call': 'pkey_mprotect',
9798               'reason': set([('vm_area_struct', 'vm_end'),
9799                              ('vm_area_struct', 'vm_start')])},
9800              {'call': 'madvise',
9801               'reason': set([('vm_area_struct', 'vm_end'),
9802                              ('vm_area_struct', 'vm_start')])},
9803              {'call': 'sched_getattr',
9804               'reason': set([('task_struct', 'personality')])},
9805              {'call': 'getrusage',
9806               'reason': set([('mm_struct', 'total_vm'),
9807                              ('task_struct', 'personality')])},
9808              {'call': 'sched_setscheduler',
9809               'reason': set([('task_struct', 'personality')])},
9810              {'call': 'setitimer',
9811               'reason': set([('task_struct', 'personality')])},
9812              {'call': 'ioprio_get',
9813               'reason': set([('task_struct', 'personality')])},
9814              {'call': 'vfork',
9815               'reason': set([('task_struct', 'personality')])},
9816              {'call': 'io_setup',
9817               'reason': set([('mm_struct', 'total_vm')])},
9818              {'call': 'mprotect',
9819               'reason': set([('vm_area_struct', 'vm_end'),
9820                              ('vm_area_struct', 'vm_start')])},
9821              {'call': 'mremap',
9822               'reason': set([('mm_struct', 'total_vm'),
9823                              ('vm_area_struct', 'vm_end'),
9824                              ('vm_area_struct', 'vm_start')])},
9825              {'call': 'io_destroy',
9826               'reason': set([('mm_struct', 'total_vm')])},
9827              {'call': 'mbind', 'reason': set([('mm_struct', 'total_vm')])},
```



```
9828              {'call': 'prctl',
9829               'reason': set([('mm_struct', 'total_vm'),
9830                              ('task_struct', 'personality'),
9831                              ('vm_area_struct', 'vm_end'),
9832                              ('vm_area_struct', 'vm_start')])},
9833              {'call': 'move_pages',
9834               'reason': set([('mm_struct', 'total_vm'),
9835                              ('task_struct', 'personality')])},
9836              {'call': 'modify_ldt',
9837               'reason': set([('mm_struct', 'total_vm')])},
9838              {'call': 'munlock',
9839               'reason': set([('vm_area_struct', 'vm_end'),
9840                              ('vm_area_struct', 'vm_start')])},
9841              {'call': 'setpriority',
9842               'reason': set([('task_struct', 'personality')])},
9843              {'call': 'mincore',
9844               'reason': set([('mm_struct', 'total_vm'),
9845                              ('vm_area_struct', 'vm_end'),
9846                              ('vm_area_struct', 'vm_start')])},
9847              {'call': 'clone',
9848               'reason': set([('task_struct', 'personality')])},
9849              {'call': 'sched_getparam',
9850               'reason': set([('task_struct', 'personality')])},
9851              {'call': 'io_cancel',
9852               'reason': set([('mm_struct', 'total_vm')])},
9853              {'call': 'mlockall',
9854               'reason': set([('vm_area_struct', 'vm_end'),
9855                              ('vm_area_struct', 'vm_start')])}],
9856  'mmap_pgoff': [{'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
9857                 {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
9858                 {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
9859                 {'call': 'remap_file_pages',
9860                  'reason': set([('file', 'f_op')])},
9861                 {'call': 'dup3', 'reason': set([('file', 'f_op')])},
9862                 {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
9863                 {'call': 'epoll_ctl', 'reason': set([('file', 'f_op')])},
9864                 {'call': 'flock', 'reason': set([('file', 'f_op')])},
9865                 {'call': 'openat', 'reason': set([('file', 'f_op')])},
9866                 {'call': 'uselib', 'reason': set([('file', 'f_op')])},
9867                 {'call': 'accept4', 'reason': set([('file', 'f_op')])},
9868                 {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
9869                 {'call': 'shmat', 'reason': set([('file', 'f_op')])},
9870                 {'call': 'socket', 'reason': set([('file', 'f_op')])},
9871                 {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
9872                 {'call': 'perf_event_open',
9873                  'reason': set([('file', 'f_op')])},
9874                 {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
9875                 {'call': 'acct', 'reason': set([('file', 'f_op')])},
9876                 {'call': 'open', 'reason': set([('file', 'f_op')])},
9877                 {'call': 'dup', 'reason': set([('file', 'f_op')])},
9878                 {'call': 'setns', 'reason': set([('file', 'f_op')])},
9879                 {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
9880                 {'call': 'swapon', 'reason': set([('file', 'f_op')])},
9881                 {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
9882                 {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
9883                 {'call': 'open_by_handle_at',
9884                  'reason': set([('file', 'f_op')])}],
9885  'modify_ldt': [{'call': 'swapoff', 'reason': set([('mm_context_t', 'ldt')])},
9886                 {'call': 'remap_file_pages',
9887                  'reason': set([('mm_context_t', 'ldt')])},
9888                 {'call': 'io_getevents',
9889                  'reason': set([('mm_context_t', 'ldt')])},
9890                 {'call': 'get_thread_area',
9891                  'reason': set([('user_desc', 'base_addr'),
9892                                 ('user_desc', 'contents'),
9893                                 ('user_desc', 'entry_number'),
```

```
                                 ('user_desc', 'limit'),
                                 ('user_desc', 'limit_in_pages'),
                                 ('user_desc', 'read_exec_only'),
                                 ('user_desc', 'seg_32bit'),
                                 ('user_desc', 'seg_not_present'),
                                 ('user_desc', 'useable')])},
                {'call': 'migrate_pages',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'shmdt', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'brk', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'get_mempolicy',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'getrusage',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'io_setup',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'mremap', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'io_destroy',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'mbind', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'prctl', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'move_pages',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'modify_ldt',
                 'reason': set([('ldt_struct', 'entries'),
                                ('mm_context_t', 'ldt'),
                                ('user_desc', 'base_addr'),
                                ('user_desc', 'contents'),
                                ('user_desc', 'entry_number'),
                                ('user_desc', 'limit'),
                                ('user_desc', 'limit_in_pages'),
                                ('user_desc', 'read_exec_only'),
                                ('user_desc', 'seg_32bit'),
                                ('user_desc', 'seg_not_present'),
                                ('user_desc', 'useable')])},
                {'call': 'mincore', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'set_thread_area',
                 'reason': set([('user_desc', 'base_addr'),
                                ('user_desc', 'contents'),
                                ('user_desc', 'entry_number'),
                                ('user_desc', 'limit'),
                                ('user_desc', 'limit_in_pages'),
                                ('user_desc', 'read_exec_only'),
                                ('user_desc', 'seg_32bit'),
                                ('user_desc', 'seg_not_present'),
                                ('user_desc', 'useable')])},
                {'call': 'io_cancel',
                 'reason': set([('mm_context_t', 'ldt')])}],
  'mount': [{'call': 'keyctl',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'msgrcv',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'ioprio_set',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'personality',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'getppid',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'mq_timedreceive',
```



```
             'reason': set([('task_struct', 'personality')])},
            {'call': 'capget',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_setaffinity',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'signal',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'semtimedop',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'umount',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'setsid',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sigaltstack',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'getitimer',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'setpgid',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'getsid',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'prlimit64',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'perf_event_open',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'getpgid',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'getpriority',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sigaction',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'ptrace',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'mount',
             'reason': set([('compat_nfs4_mount_data_v1', 'auth_flavours'),
                            ('compat_nfs4_mount_data_v1', 'host_addr')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'getrusage',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'setitimer',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'ioprio_get',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
            {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
```

```
10026            {'call': 'move_pages',
10027             'reason': set([('task_struct', 'personality')])},
10028            {'call': 'setpriority',
10029             'reason': set([('task_struct', 'personality')])},
10030            {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
10031            {'call': 'sched_getparam',
10032             'reason': set([('task_struct', 'personality')])}],
10033  'mprotect': [{'call': 'keyctl',
10034                'reason': set([('task_struct', 'mm'),
10035                               ('task_struct', 'personality')])},
10036               {'call': 'rt_sigtimedwait',
10037                'reason': set([('task_struct', 'mm'),
10038                               ('task_struct', 'personality')])},
10039               {'call': 'msgrcv',
10040                'reason': set([('task_struct', 'mm'),
10041                               ('task_struct', 'personality')])},
10042               {'call': 'kill',
10043                'reason': set([('task_struct', 'mm'),
10044                               ('task_struct', 'personality')])},
10045               {'call': 'sched_getaffinity',
10046                'reason': set([('task_struct', 'mm'),
10047                               ('task_struct', 'personality')])},
10048               {'call': 'sched_setparam',
10049                'reason': set([('task_struct', 'mm'),
10050                               ('task_struct', 'personality')])},
10051               {'call': 'ioprio_set',
10052                'reason': set([('task_struct', 'mm'),
10053                               ('task_struct', 'personality')])},
10054               {'call': 'personality',
10055                'reason': set([('task_struct', 'personality')])},
10056               {'call': 'remap_file_pages',
10057                'reason': set([('vm_area_struct', 'vm_end'),
10058                               ('vm_area_struct', 'vm_flags'),
10059                               ('vm_area_struct', 'vm_start')])},
10060               {'call': 'getppid',
10061                'reason': set([('task_struct', 'mm'),
10062                               ('task_struct', 'personality')])},
10063               {'call': 'mq_timedreceive',
10064                'reason': set([('task_struct', 'mm'),
10065                               ('task_struct', 'personality')])},
10066               {'call': 'capget',
10067                'reason': set([('task_struct', 'mm'),
10068                               ('task_struct', 'personality')])},
10069               {'call': 'sched_setaffinity',
10070                'reason': set([('task_struct', 'mm'),
10071                               ('task_struct', 'personality')])},
10072               {'call': 'signal',
10073                'reason': set([('task_struct', 'mm'),
10074                               ('task_struct', 'personality')])},
10075               {'call': 'semtimedop',
10076                'reason': set([('task_struct', 'mm'),
10077                               ('task_struct', 'personality')])},
10078               {'call': 'umount',
10079                'reason': set([('task_struct', 'mm'),
10080                               ('task_struct', 'personality')])},
10081               {'call': 'sched_rr_get_interval',
10082                'reason': set([('task_struct', 'mm'),
10083                               ('task_struct', 'personality')])},
10084               {'call': 'rt_sigprocmask',
10085                'reason': set([('task_struct', 'mm'),
10086                               ('task_struct', 'personality')])},
10087               {'call': 'setsid',
10088                'reason': set([('task_struct', 'mm'),
10089                               ('task_struct', 'personality')])},
10090               {'call': 'sigaltstack',
10091                'reason': set([('task_struct', 'mm'),
```



```
10092                               ('task_struct', 'personality')])},
10093               {'call': 'sched_setattr',
10094                'reason': set([('task_struct', 'mm'),
10095                               ('task_struct', 'personality')])},
10096               {'call': 'migrate_pages',
10097                'reason': set([('task_struct', 'mm'),
10098                               ('task_struct', 'personality')])},
10099               {'call': 'getitimer',
10100                'reason': set([('task_struct', 'mm'),
10101                               ('task_struct', 'personality')])},
10102               {'call': 'setpgid',
10103                'reason': set([('task_struct', 'mm'),
10104                               ('task_struct', 'personality')])},
10105               {'call': 'getsid',
10106                'reason': set([('task_struct', 'mm'),
10107                               ('task_struct', 'personality')])},
10108               {'call': 'prlimit64',
10109                'reason': set([('task_struct', 'mm'),
10110                               ('task_struct', 'personality')])},
10111               {'call': 'perf_event_open',
10112                'reason': set([('task_struct', 'mm'),
10113                               ('task_struct', 'personality')])},
10114               {'call': 'shmdt',
10115                'reason': set([('vm_area_struct', 'vm_end'),
10116                               ('vm_area_struct', 'vm_flags'),
10117                               ('vm_area_struct', 'vm_start')])},
10118               {'call': 'rt_sigaction',
10119                'reason': set([('task_struct', 'mm'),
10120                               ('task_struct', 'personality')])},
10121               {'call': 'getpgid',
10122                'reason': set([('task_struct', 'mm'),
10123                               ('task_struct', 'personality')])},
10124               {'call': 'brk',
10125                'reason': set([('vm_area_struct', 'vm_end'),
10126                               ('vm_area_struct', 'vm_flags'),
10127                               ('vm_area_struct', 'vm_start')])},
10128               {'call': 'getpriority',
10129                'reason': set([('task_struct', 'mm'),
10130                               ('task_struct', 'personality')])},
10131               {'call': 'sigaction',
10132                'reason': set([('task_struct', 'mm'),
10133                               ('task_struct', 'personality')])},
10134               {'call': 'setns',
10135                'reason': set([('task_struct', 'mm'),
10136                               ('task_struct', 'personality')])},
10137               {'call': 'fork',
10138                'reason': set([('task_struct', 'mm'),
10139                               ('task_struct', 'personality')])},
10140               {'call': 'get_mempolicy',
10141                'reason': set([('vm_area_struct', 'vm_end'),
10142                               ('vm_area_struct', 'vm_flags'),
10143                               ('vm_area_struct', 'vm_start')])},
10144               {'call': 'get_robust_list',
10145                'reason': set([('task_struct', 'mm'),
10146                               ('task_struct', 'personality')])},
10147               {'call': 'mq_timedsend',
10148                'reason': set([('task_struct', 'mm'),
10149                               ('task_struct', 'personality')])},
10150               {'call': 'sched_getscheduler',
10151                'reason': set([('task_struct', 'mm'),
10152                               ('task_struct', 'personality')])},
10153               {'call': 'ptrace',
10154                'reason': set([('task_struct', 'mm'),
10155                               ('task_struct', 'personality')])},
10156               {'call': 'munlockall',
10157                'reason': set([('vm_area_struct', 'vm_end'),
```

```
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'pkey_mprotect',
             'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'madvise',
             'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'getrusage',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'setitimer',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'ioprio_get',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'vfork',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'mprotect',
             'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'mremap',
             'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'prctl',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality'),
                                ('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'move_pages',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'munlock',
             'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'setpriority',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'mincore',
             'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
            {'call': 'clone',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'sched_getparam',
             'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
            {'call': 'mlockall',
             'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])}],
```



```
 'mq_getsetattr': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
                   {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
                   {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
                   {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
                   {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
                   {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
                   {'call': 'readahead', 'reason': set([('fd', 'file')])},
                   {'call': 'getdents', 'reason': set([('fd', 'file')])},
                   {'call': 'writev', 'reason': set([('fd', 'file')])},
                   {'call': 'preadv64', 'reason': set([('fd', 'file')])},
                   {'call': 'fchmod', 'reason': set([('fd', 'file')])},
                   {'call': 'pread64', 'reason': set([('fd', 'file')])},
                   {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
                   {'call': 'memfd_create',
                    'reason': set([('file', 'f_op')])},
                   {'call': 'remap_file_pages',
                    'reason': set([('file', 'f_op')])},
                   {'call': 'dup3', 'reason': set([('file', 'f_op')])},
                   {'call': 'read', 'reason': set([('fd', 'file')])},
                   {'call': 'fchown', 'reason': set([('fd', 'file')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('fd', 'file'), ('mq_attr', 'mq_flags')])},
                   {'call': 'utime', 'reason': set([('fd', 'file')])},
                   {'call': 'fsync', 'reason': set([('fd', 'file')])},
                   {'call': 'bpf', 'reason': set([('fd', 'file')])},
                   {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
                   {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
                   {'call': 'sendto', 'reason': set([('fd', 'file')])},
                   {'call': 'epoll_create1',
                    'reason': set([('file', 'f_op')])},
                   {'call': 'tee', 'reason': set([('fd', 'file')])},
                   {'call': 'sync_file_range',
                    'reason': set([('fd', 'file')])},
                   {'call': 'lseek', 'reason': set([('fd', 'file')])},
                   {'call': 'connect', 'reason': set([('fd', 'file')])},
                   {'call': 'getsockname', 'reason': set([('fd', 'file')])},
                   {'call': 'epoll_ctl',
                    'reason': set([('fd', 'file'), ('file', 'f_op')])},
                   {'call': 'flock',
                    'reason': set([('fd', 'file'), ('file', 'f_op')])},
                   {'call': 'pwritev', 'reason': set([('fd', 'file')])},
                   {'call': 'fchdir', 'reason': set([('fd', 'file')])},
                   {'call': 'openat', 'reason': set([('file', 'f_op')])},
                   {'call': 'uselib', 'reason': set([('file', 'f_op')])},
                   {'call': 'accept4',
                    'reason': set([('fd', 'file'), ('file', 'f_op')])},
                   {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
                   {'call': 'inotify_rm_watch',
                    'reason': set([('fd', 'file')])},
                   {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
                   {'call': 'utimensat', 'reason': set([('fd', 'file')])},
                   {'call': 'inotify_add_watch',
                    'reason': set([('fd', 'file')])},
                   {'call': 'preadv2', 'reason': set([('fd', 'file')])},
                   {'call': 'splice', 'reason': set([('fd', 'file')])},
                   {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
                   {'call': 'preadv', 'reason': set([('fd', 'file')])},
                   {'call': 'getpeername', 'reason': set([('fd', 'file')])},
                   {'call': 'shmat', 'reason': set([('file', 'f_op')])},
                   {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
                   {'call': 'socket', 'reason': set([('file', 'f_op')])},
                   {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
                   {'call': 'fcntl', 'reason': set([('fd', 'file')])},
                   {'call': 'ioctl', 'reason': set([('fd', 'file')])},
                   {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
                   {'call': 'perf_event_open',
```

```
10290                     'reason': set([('fd', 'file'), ('file', 'f_op')])},
10291                    {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
10292                    {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
10293                    {'call': 'futimesat', 'reason': set([('fd', 'file')])},
10294                    {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
10295                    {'call': 'shutdown', 'reason': set([('fd', 'file')])},
10296                    {'call': 'acct', 'reason': set([('file', 'f_op')])},
10297                    {'call': 'open', 'reason': set([('file', 'f_op')])},
10298                    {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
10299                    {'call': 'mq_getsetattr',
10300                     'reason': set([('fd', 'file'), ('mq_attr', 'mq_flags')])},
10301                    {'call': 'dup', 'reason': set([('file', 'f_op')])},
10302                    {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
10303                    {'call': 'setns', 'reason': set([('file', 'f_op')])},
10304                    {'call': 'getdents64', 'reason': set([('fd', 'file')])},
10305                    {'call': 'listen', 'reason': set([('fd', 'file')])},
10306                    {'call': 'copy_file_range',
10307                     'reason': set([('fd', 'file')])},
10308                    {'call': 'mq_timedsend',
10309                     'reason': set([('fd', 'file'), ('mq_attr', 'mq_flags')])},
10310                    {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
10311                    {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
10312                    {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
10313                    {'call': 'swapon', 'reason': set([('file', 'f_op')])},
10314                    {'call': 'fallocate', 'reason': set([('fd', 'file')])},
10315                    {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
10316                    {'call': 'llseek', 'reason': set([('fd', 'file')])},
10317                    {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
10318                    {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
10319                    {'call': 'readv', 'reason': set([('fd', 'file')])},
10320                    {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
10321                    {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
10322                    {'call': 'write', 'reason': set([('fd', 'file')])},
10323                    {'call': 'mq_notify',
10324                     'reason': set([('fd', 'file'), ('mq_attr', 'mq_flags')])},
10325                    {'call': 'sendfile', 'reason': set([('fd', 'file')])},
10326                    {'call': 'mq_open',
10327                     'reason': set([('file', 'f_op'),
10328                                    ('mq_attr', 'mq_flags')])},
10329                    {'call': 'open_by_handle_at',
10330                     'reason': set([('file', 'f_op')])},
10331                    {'call': 'bind', 'reason': set([('fd', 'file')])},
10332                    {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
10333                    {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
10334  'mq_notify': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
10335                 {'call': 'rt_sigtimedwait',
10336                  'reason': set([('sigval', 'sival_ptr')])},
10337                 {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
10338                 {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
10339                 {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
10340                 {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
10341                 {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
10342                 {'call': 'readahead', 'reason': set([('fd', 'file')])},
10343                 {'call': 'getdents', 'reason': set([('fd', 'file')])},
10344                 {'call': 'writev', 'reason': set([('fd', 'file')])},
10345                 {'call': 'preadv64', 'reason': set([('fd', 'file')])},
10346                 {'call': 'fchmod', 'reason': set([('fd', 'file')])},
10347                 {'call': 'pread64', 'reason': set([('fd', 'file')])},
10348                 {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
10349                 {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
10350                 {'call': 'remap_file_pages',
10351                  'reason': set([('file', 'f_op')])},
10352                 {'call': 'dup3', 'reason': set([('file', 'f_op')])},
10353                 {'call': 'read', 'reason': set([('fd', 'file')])},
10354                 {'call': 'fchown', 'reason': set([('fd', 'file')])},
10355                 {'call': 'mq_timedreceive',
```



```
10356                  'reason': set([('fd', 'file'),
10357                                 ('mqueue_inode_info', 'notify_owner'),
10358                                 ('sigevent', 'sigev_notify'),
10359                                 ('sigevent', 'sigev_signo'),
10360                                 ('sigval', 'sival_ptr')])},
10361                 {'call': 'utime', 'reason': set([('fd', 'file')])},
10362                 {'call': 'fsync', 'reason': set([('fd', 'file')])},
10363                 {'call': 'bpf', 'reason': set([('fd', 'file')])},
10364                 {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
10365                 {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
10366                 {'call': 'timer_create',
10367                  'reason': set([('sigevent', 'sigev_notify'),
10368                                 ('sigevent', 'sigev_signo'),
10369                                 ('sigval', 'sival_ptr')])},
10370                 {'call': 'sendto', 'reason': set([('fd', 'file')])},
10371                 {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
10372                 {'call': 'tee', 'reason': set([('fd', 'file')])},
10373                 {'call': 'rt_sigqueueinfo',
10374                  'reason': set([('sigval', 'sival_ptr')])},
10375                 {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
10376                 {'call': 'lseek', 'reason': set([('fd', 'file')])},
10377                 {'call': 'connect', 'reason': set([('fd', 'file')])},
10378                 {'call': 'getsockname', 'reason': set([('fd', 'file')])},
10379                 {'call': 'epoll_ctl',
10380                  'reason': set([('fd', 'file'), ('file', 'f_op')])},
10381                 {'call': 'flock',
10382                  'reason': set([('fd', 'file'), ('file', 'f_op')])},
10383                 {'call': 'pwritev', 'reason': set([('fd', 'file')])},
10384                 {'call': 'fchdir', 'reason': set([('fd', 'file')])},
10385                 {'call': 'tgkill', 'reason': set([('sigval', 'sival_ptr')])},
10386                 {'call': 'openat', 'reason': set([('file', 'f_op')])},
10387                 {'call': 'uselib', 'reason': set([('file', 'f_op')])},
10388                 {'call': 'accept4',
10389                  'reason': set([('fd', 'file'), ('file', 'f_op')])},
10390                 {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
10391                 {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
10392                 {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
10393                 {'call': 'utimensat', 'reason': set([('fd', 'file')])},
10394                 {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
10395                 {'call': 'preadv2', 'reason': set([('fd', 'file')])},
10396                 {'call': 'splice', 'reason': set([('fd', 'file')])},
10397                 {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
10398                 {'call': 'preadv', 'reason': set([('fd', 'file')])},
10399                 {'call': 'getpeername', 'reason': set([('fd', 'file')])},
10400                 {'call': 'shmat', 'reason': set([('file', 'f_op')])},
10401                 {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
10402                 {'call': 'socket', 'reason': set([('file', 'f_op')])},
10403                 {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
10404                 {'call': 'fcntl', 'reason': set([('fd', 'file')])},
10405                 {'call': 'ioctl', 'reason': set([('fd', 'file')])},
10406                 {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
10407                 {'call': 'perf_event_open',
10408                  'reason': set([('fd', 'file'), ('file', 'f_op')])},
10409                 {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
10410                 {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
10411                 {'call': 'futimesat', 'reason': set([('fd', 'file')])},
10412                 {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
10413                 {'call': 'shutdown', 'reason': set([('fd', 'file')])},
10414                 {'call': 'acct', 'reason': set([('file', 'f_op')])},
10415                 {'call': 'open', 'reason': set([('file', 'f_op')])},
10416                 {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
10417                 {'call': 'rt_tgsigqueueinfo',
10418                  'reason': set([('sigval', 'sival_ptr')])},
10419                 {'call': 'mq_getsetattr',
10420                  'reason': set([('fd', 'file'),
10421                                 ('mqueue_inode_info', 'notify_owner'),
```

```
10422                                 (’sigevent’, ’sigev_notify’),
10423                                 (’sigevent’, ’sigev_signo’),
10424                                 (’sigval’, ’sival_ptr’)])},
10425                 {’call’: ’dup’, ’reason’: set([(’file’, ’f_op’)])},
10426                 {’call’: ’fdatasync’, ’reason’: set([(’fd’, ’file’)])},
10427                 {’call’: ’setns’, ’reason’: set([(’file’, ’f_op’)])},
10428                 {’call’: ’getdents64’, ’reason’: set([(’fd’, ’file’)])},
10429                 {’call’: ’listen’, ’reason’: set([(’fd’, ’file’)])},
10430                 {’call’: ’copy_file_range’, ’reason’: set([(’fd’, ’file’)])},
10431                 {’call’: ’mq_timedsend’,
10432                  ’reason’: set([(’fd’, ’file’),
10433                                 (’mqueue_inode_info’, ’notify_owner’),
10434                                 (’sigevent’, ’sigev_notify’),
10435                                 (’sigevent’, ’sigev_signo’),
10436                                 (’sigval’, ’sival_ptr’)])},
10437                 {’call’: ’fgetxattr’, ’reason’: set([(’fd’, ’file’)])},
10438                 {’call’: ’shmctl’, ’reason’: set([(’file’, ’f_op’)])},
10439                 {’call’: ’fcntl64’, ’reason’: set([(’fd’, ’file’)])},
10440                 {’call’: ’swapon’, ’reason’: set([(’file’, ’f_op’)])},
10441                 {’call’: ’fallocate’, ’reason’: set([(’fd’, ’file’)])},
10442                 {’call’: ’epoll_wait’, ’reason’: set([(’fd’, ’file’)])},
10443                 {’call’: ’llseek’, ’reason’: set([(’fd’, ’file’)])},
10444                 {’call’: ’mmap_pgoff’, ’reason’: set([(’file’, ’f_op’)])},
10445                 {’call’: ’preadv64v2’, ’reason’: set([(’fd’, ’file’)])},
10446                 {’call’: ’readv’, ’reason’: set([(’fd’, ’file’)])},
10447                 {’call’: ’fstatfs’, ’reason’: set([(’fd’, ’file’)])},
10448                 {’call’: ’fstatfs64’, ’reason’: set([(’fd’, ’file’)])},
10449                 {’call’: ’rt_sigreturn’,
10450                  ’reason’: set([(’sigval’, ’sival_ptr’)])},
10451                 {’call’: ’write’, ’reason’: set([(’fd’, ’file’)])},
10452                 {’call’: ’tkill’, ’reason’: set([(’sigval’, ’sival_ptr’)])},
10453                 {’call’: ’mq_notify’,
10454                  ’reason’: set([(’fd’, ’file’),
10455                                 (’mqueue_inode_info’, ’notify_owner’),
10456                                 (’sigevent’, ’sigev_notify’),
10457                                 (’sigevent’, ’sigev_signo’),
10458                                 (’sigval’, ’sival_ptr’),
10459                                 (’sk_buff’, ’data’)])},
10460                 {’call’: ’sendfile’, ’reason’: set([(’fd’, ’file’)])},
10461                 {’call’: ’mq_open’, ’reason’: set([(’file’, ’f_op’)])},
10462                 {’call’: ’open_by_handle_at’,
10463                  ’reason’: set([(’file’, ’f_op’)])},
10464                 {’call’: ’bind’, ’reason’: set([(’fd’, ’file’)])},
10465                 {’call’: ’flistxattr’, ’reason’: set([(’fd’, ’file’)])},
10466                 {’call’: ’sendfile64’, ’reason’: set([(’fd’, ’file’)])}],
10467  ’mq_open’: [{’call’: ’sysfs’, ’reason’: set([(’filename’, ’name’)])},
10468               {’call’: ’eventfd2’, ’reason’: set([(’path’, ’dentry’)])},
10469               {’call’: ’mq_unlink’, ’reason’: set([(’filename’, ’name’)])},
10470               {’call’: ’swapoff’,
10471                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10472               {’call’: ’pivot_root’, ’reason’: set([(’path’, ’dentry’)])},
10473               {’call’: ’memfd_create’, ’reason’: set([(’path’, ’dentry’)])},
10474               {’call’: ’remap_file_pages’,
10475                ’reason’: set([(’path’, ’dentry’)])},
10476               {’call’: ’dup3’, ’reason’: set([(’path’, ’dentry’)])},
10477               {’call’: ’unshare’, ’reason’: set([(’path’, ’dentry’)])},
10478               {’call’: ’epoll_create1’, ’reason’: set([(’path’, ’dentry’)])},
10479               {’call’: ’epoll_ctl’, ’reason’: set([(’path’, ’dentry’)])},
10480               {’call’: ’flock’, ’reason’: set([(’path’, ’dentry’)])},
10481               {’call’: ’openat’,
10482                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10483               {’call’: ’lookup_dcookie’, ’reason’: set([(’path’, ’dentry’)])},
10484               {’call’: ’uselib’,
10485                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10486               {’call’: ’renameat2’, ’reason’: set([(’filename’, ’name’)])},
10487               {’call’: ’accept4’, ’reason’: set([(’path’, ’dentry’)])},
```



```
10488               {’call’: ’socketpair’, ’reason’: set([(’path’, ’dentry’)])},
10489               {’call’: ’getcwd’, ’reason’: set([(’path’, ’dentry’)])},
10490               {’call’: ’shmat’, ’reason’: set([(’path’, ’dentry’)])},
10491               {’call’: ’socket’, ’reason’: set([(’path’, ’dentry’)])},
10492               {’call’: ’symlinkat’, ’reason’: set([(’filename’, ’name’)])},
10493               {’call’: ’pipe2’, ’reason’: set([(’path’, ’dentry’)])},
10494               {’call’: ’perf_event_open’, ’reason’: set([(’path’, ’dentry’)])},
10495               {’call’: ’shmdt’, ’reason’: set([(’path’, ’dentry’)])},
10496               {’call’: ’quotactl’,
10497                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10498               {’call’: ’acct’,
10499                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10500               {’call’: ’open’,
10501                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10502               {’call’: ’unlink’, ’reason’: set([(’filename’, ’name’)])},
10503               {’call’: ’rmdir’, ’reason’: set([(’filename’, ’name’)])},
10504               {’call’: ’dup’, ’reason’: set([(’path’, ’dentry’)])},
10505               {’call’: ’setns’, ’reason’: set([(’path’, ’dentry’)])},
10506               {’call’: ’shmctl’, ’reason’: set([(’path’, ’dentry’)])},
10507               {’call’: ’swapon’,
10508                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10509               {’call’: ’mmap_pgoff’, ’reason’: set([(’path’, ’dentry’)])},
10510               {’call’: ’mq_open’,
10511                ’reason’: set([(’filename’, ’name’), (’path’, ’dentry’)])},
10512               {’call’: ’unlinkat’, ’reason’: set([(’filename’, ’name’)])},
10513               {’call’: ’open_by_handle_at’,
10514                ’reason’: set([(’path’, ’dentry’)])}],
10515  ’mq_timedreceive’: [{’call’: ’syncfs’, ’reason’: set([(’fd’, ’file’)])},
10516                      {’call’: ’vmsplice’, ’reason’: set([(’fd’, ’file’)])},
10517                      {’call’: ’msgrcv’, ’reason’: set([(’msg_msg’, ’m_ts’)])},
10518                      {’call’: ’eventfd2’,
10519                       ’reason’: set([(’file’, ’f_flags’),
10520                                      (’file’, ’f_mode’),
10521                                      (’file’, ’f_op’)])},
10522                      {’call’: ’pwritev64’, ’reason’: set([(’fd’, ’file’)])},
10523                      {’call’: ’swapoff’,
10524                       ’reason’: set([(’file’, ’f_flags’),
10525                                      (’file’, ’f_mode’),
10526                                      (’file’, ’f_op’)])},
10527                      {’call’: ’fremovexattr’,
10528                       ’reason’: set([(’fd’, ’file’)])},
10529                      {’call’: ’readahead’, ’reason’: set([(’fd’, ’file’)])},
10530                      {’call’: ’getdents’, ’reason’: set([(’fd’, ’file’)])},
10531                      {’call’: ’writev’, ’reason’: set([(’fd’, ’file’)])},
10532                      {’call’: ’preadv64’, ’reason’: set([(’fd’, ’file’)])},
10533                      {’call’: ’fchmod’, ’reason’: set([(’fd’, ’file’)])},
10534                      {’call’: ’pread64’, ’reason’: set([(’fd’, ’file’)])},
10535                      {’call’: ’signalfd4’, ’reason’: set([(’fd’, ’file’)])},
10536                      {’call’: ’memfd_create’,
10537                       ’reason’: set([(’file’, ’f_flags’),
10538                                      (’file’, ’f_mode’),
10539                                      (’file’, ’f_op’)])},
10540                      {’call’: ’remap_file_pages’,
10541                       ’reason’: set([(’file’, ’f_flags’),
10542                                      (’file’, ’f_mode’),
10543                                      (’file’, ’f_op’)])},
10544                      {’call’: ’dup3’,
10545                       ’reason’: set([(’file’, ’f_flags’),
10546                                      (’file’, ’f_mode’),
10547                                      (’file’, ’f_op’)])},
10548                      {’call’: ’read’, ’reason’: set([(’fd’, ’file’)])},
10549                      {’call’: ’fchown’, ’reason’: set([(’fd’, ’file’)])},
10550                      {’call’: ’mq_timedreceive’,
10551                       ’reason’: set([(’fd’, ’file’),
10552                                      (’mq_attr’, ’mq_curmsgs’),
10553                                      (’mq_attr’, ’mq_msgsize’),
```

```
10554                                   ('mqueue_inode_info', 'node_cache'),
10555                                   ('msg_msg', 'm_ts')])},
10556                  {'call': 'utime', 'reason': set([('fd', 'file')])},
10557                  {'call': 'fsync', 'reason': set([('fd', 'file')])},
10558                  {'call': 'bpf', 'reason': set([('fd', 'file')])},
10559                  {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
10560                  {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
10561                  {'call': 'sendto', 'reason': set([('fd', 'file')])},
10562                  {'call': 'epoll_create1',
10563                   'reason': set([('file', 'f_flags'),
10564                                  ('file', 'f_mode'),
10565                                  ('file', 'f_op')])},
10566                  {'call': 'tee', 'reason': set([('fd', 'file')])},
10567                  {'call': 'sync_file_range',
10568                   'reason': set([('fd', 'file')])},
10569                  {'call': 'lseek', 'reason': set([('fd', 'file')])},
10570                  {'call': 'connect', 'reason': set([('fd', 'file')])},
10571                  {'call': 'getsockname', 'reason': set([('fd', 'file')])},
10572                  {'call': 'epoll_ctl',
10573                   'reason': set([('fd', 'file'),
10574                                  ('file', 'f_flags'),
10575                                  ('file', 'f_mode'),
10576                                  ('file', 'f_op')])},
10577                  {'call': 'flock',
10578                   'reason': set([('fd', 'file'),
10579                                  ('file', 'f_flags'),
10580                                  ('file', 'f_mode'),
10581                                  ('file', 'f_op')])},
10582                  {'call': 'pwritev', 'reason': set([('fd', 'file')])},
10583                  {'call': 'fchdir', 'reason': set([('fd', 'file')])},
10584                  {'call': 'openat',
10585                   'reason': set([('file', 'f_flags'),
10586                                  ('file', 'f_mode'),
10587                                  ('file', 'f_op')])},
10588                  {'call': 'uselib',
10589                   'reason': set([('file', 'f_flags'),
10590                                  ('file', 'f_mode'),
10591                                  ('file', 'f_op')])},
10592                  {'call': 'accept4',
10593                   'reason': set([('fd', 'file'),
10594                                  ('file', 'f_flags'),
10595                                  ('file', 'f_mode'),
10596                                  ('file', 'f_op')])},
10597                  {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
10598                  {'call': 'inotify_rm_watch',
10599                   'reason': set([('fd', 'file')])},
10600                  {'call': 'socketpair',
10601                   'reason': set([('file', 'f_flags'),
10602                                  ('file', 'f_mode'),
10603                                  ('file', 'f_op')])},
10604                  {'call': 'utimensat', 'reason': set([('fd', 'file')])},
10605                  {'call': 'inotify_add_watch',
10606                   'reason': set([('fd', 'file')])},
10607                  {'call': 'preadv2', 'reason': set([('fd', 'file')])},
10608                  {'call': 'splice', 'reason': set([('fd', 'file')])},
10609                  {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
10610                  {'call': 'preadv', 'reason': set([('fd', 'file')])},
10611                  {'call': 'getpeername', 'reason': set([('fd', 'file')])},
10612                  {'call': 'shmat',
10613                   'reason': set([('file', 'f_flags'),
10614                                  ('file', 'f_mode'),
10615                                  ('file', 'f_op')])},
10616                  {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
10617                  {'call': 'socket',
10618                   'reason': set([('file', 'f_flags'),
10619                                  ('file', 'f_mode'),
```



```
10620                                   ('file', 'f_op')])},
10621                  {'call': 'pipe2',
10622                   'reason': set([('file', 'f_flags'),
10623                                  ('file', 'f_mode'),
10624                                  ('file', 'f_op')])},
10625                  {'call': 'fcntl', 'reason': set([('fd', 'file')])},
10626                  {'call': 'ioctl', 'reason': set([('fd', 'file')])},
10627                  {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
10628                  {'call': 'perf_event_open',
10629                   'reason': set([('fd', 'file'),
10630                                  ('file', 'f_flags'),
10631                                  ('file', 'f_mode'),
10632                                  ('file', 'f_op')])},
10633                  {'call': 'shmdt',
10634                   'reason': set([('file', 'f_flags'),
10635                                  ('file', 'f_mode'),
10636                                  ('file', 'f_op')])},
10637                  {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
10638                  {'call': 'futimesat', 'reason': set([('fd', 'file')])},
10639                  {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
10640                  {'call': 'shutdown', 'reason': set([('fd', 'file')])},
10641                  {'call': 'acct',
10642                   'reason': set([('file', 'f_flags'),
10643                                  ('file', 'f_mode'),
10644                                  ('file', 'f_op')])},
10645                  {'call': 'open',
10646                   'reason': set([('file', 'f_flags'),
10647                                  ('file', 'f_mode'),
10648                                  ('file', 'f_op')])},
10649                  {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
10650                  {'call': 'mq_getsetattr',
10651                   'reason': set([('fd', 'file'),
10652                                  ('file', 'f_flags'),
10653                                  ('mq_attr', 'mq_curmsgs'),
10654                                  ('mq_attr', 'mq_msgsize'),
10655                                  ('mqueue_inode_info', 'node_cache')])},
10656                  {'call': 'dup',
10657                   'reason': set([('file', 'f_flags'),
10658                                  ('file', 'f_mode'),
10659                                  ('file', 'f_op')])},
10660                  {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
10661                  {'call': 'setns',
10662                   'reason': set([('file', 'f_flags'),
10663                                  ('file', 'f_mode'),
10664                                  ('file', 'f_op')])},
10665                  {'call': 'getdents64', 'reason': set([('fd', 'file')])},
10666                  {'call': 'listen', 'reason': set([('fd', 'file')])},
10667                  {'call': 'copy_file_range',
10668                   'reason': set([('fd', 'file')])},
10669                  {'call': 'mq_timedsend',
10670                   'reason': set([('fd', 'file'),
10671                                  ('mq_attr', 'mq_curmsgs'),
10672                                  ('mq_attr', 'mq_msgsize'),
10673                                  ('mqueue_inode_info', 'node_cache'),
10674                                  ('msg_msg', 'm_ts')])},
10675                  {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
10676                  {'call': 'shmctl',
10677                   'reason': set([('file', 'f_flags'),
10678                                  ('file', 'f_mode'),
10679                                  ('file', 'f_op')])},
10680                  {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
10681                  {'call': 'swapon',
10682                   'reason': set([('file', 'f_flags'),
10683                                  ('file', 'f_mode'),
10684                                  ('file', 'f_op')])},
10685                  {'call': 'fallocate', 'reason': set([('fd', 'file')])},
```

```
10686                     {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
10687                     {'call': 'llseek', 'reason': set([('fd', 'file')])},
10688                     {'call': 'mmap_pgoff',
10689                      'reason': set([('file', 'f_flags'),
10690                                     ('file', 'f_mode'),
10691                                     ('file', 'f_op')])},
10692                     {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
10693                     {'call': 'readv', 'reason': set([('fd', 'file')])},
10694                     {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
10695                     {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
10696                     {'call': 'msgsnd', 'reason': set([('msg_msg', 'm_ts')])},
10697                     {'call': 'write', 'reason': set([('fd', 'file')])},
10698                     {'call': 'mq_notify',
10699                      'reason': set([('fd', 'file'),
10700                                     ('mq_attr', 'mq_curmsgs'),
10701                                     ('mq_attr', 'mq_msgsize'),
10702                                     ('mqueue_inode_info', 'node_cache')])},
10703                     {'call': 'sendfile', 'reason': set([('fd', 'file')])},
10704                     {'call': 'mq_open',
10705                      'reason': set([('file', 'f_flags'),
10706                                     ('file', 'f_mode'),
10707                                     ('file', 'f_op'),
10708                                     ('mq_attr', 'mq_curmsgs'),
10709                                     ('mq_attr', 'mq_msgsize')])},
10710                     {'call': 'open_by_handle_at',
10711                      'reason': set([('file', 'f_flags'),
10712                                     ('file', 'f_mode'),
10713                                     ('file', 'f_op')])},
10714                     {'call': 'bind', 'reason': set([('fd', 'file')])},
10715                     {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
10716                     {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
10717  'mq_timedsend': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
10718                   {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
10719                   {'call': 'eventfd2',
10720                    'reason': set([('file', 'f_flags'),
10721                                   ('file', 'f_mode'),
10722                                   ('file', 'f_op')])},
10723                   {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
10724                   {'call': 'swapoff',
10725                    'reason': set([('file', 'f_flags'),
10726                                   ('file', 'f_mode'),
10727                                   ('file', 'f_op')])},
10728                   {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
10729                   {'call': 'readahead', 'reason': set([('fd', 'file')])},
10730                   {'call': 'getdents', 'reason': set([('fd', 'file')])},
10731                   {'call': 'writev', 'reason': set([('fd', 'file')])},
10732                   {'call': 'preadv64', 'reason': set([('fd', 'file')])},
10733                   {'call': 'fchmod', 'reason': set([('fd', 'file')])},
10734                   {'call': 'pread64', 'reason': set([('fd', 'file')])},
10735                   {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
10736                   {'call': 'memfd_create',
10737                    'reason': set([('file', 'f_flags'),
10738                                   ('file', 'f_mode'),
10739                                   ('file', 'f_op')])},
10740                   {'call': 'remap_file_pages',
10741                    'reason': set([('file', 'f_flags'),
10742                                   ('file', 'f_mode'),
10743                                   ('file', 'f_op')])},
10744                   {'call': 'dup3',
10745                    'reason': set([('file', 'f_flags'),
10746                                   ('file', 'f_mode'),
10747                                   ('file', 'f_op')])},
10748                   {'call': 'read', 'reason': set([('fd', 'file')])},
10749                   {'call': 'fchown', 'reason': set([('fd', 'file')])},
10750                   {'call': 'mq_timedreceive',
10751                    'reason': set([('fd', 'file'),
```



```
10752                                   ('mq_attr', 'mq_curmsgs'),
10753                                   ('mq_attr', 'mq_maxmsg'),
10754                                   ('mq_attr', 'mq_msgsize'),
10755                                   ('mqueue_inode_info', 'node_cache')])},
10756                   {'call': 'utime', 'reason': set([('fd', 'file')])},
10757                   {'call': 'fsync', 'reason': set([('fd', 'file')])},
10758                   {'call': 'bpf', 'reason': set([('fd', 'file')])},
10759                   {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
10760                   {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
10761                   {'call': 'sendto', 'reason': set([('fd', 'file')])},
10762                   {'call': 'epoll_create1',
10763                    'reason': set([('file', 'f_flags'),
10764                                   ('file', 'f_mode'),
10765                                   ('file', 'f_op')])},
10766                   {'call': 'tee', 'reason': set([('fd', 'file')])},
10767                   {'call': 'sync_file_range',
10768                    'reason': set([('fd', 'file')])},
10769                   {'call': 'lseek', 'reason': set([('fd', 'file')])},
10770                   {'call': 'connect', 'reason': set([('fd', 'file')])},
10771                   {'call': 'getsockname', 'reason': set([('fd', 'file')])},
10772                   {'call': 'epoll_ctl',
10773                    'reason': set([('fd', 'file'),
10774                                   ('file', 'f_flags'),
10775                                   ('file', 'f_mode'),
10776                                   ('file', 'f_op')])},
10777                   {'call': 'flock',
10778                    'reason': set([('fd', 'file'),
10779                                   ('file', 'f_flags'),
10780                                   ('file', 'f_mode'),
10781                                   ('file', 'f_op')])},
10782                   {'call': 'pwritev', 'reason': set([('fd', 'file')])},
10783                   {'call': 'fchdir', 'reason': set([('fd', 'file')])},
10784                   {'call': 'openat',
10785                    'reason': set([('file', 'f_flags'),
10786                                   ('file', 'f_mode'),
10787                                   ('file', 'f_op')])},
10788                   {'call': 'uselib',
10789                    'reason': set([('file', 'f_flags'),
10790                                   ('file', 'f_mode'),
10791                                   ('file', 'f_op')])},
10792                   {'call': 'accept4',
10793                    'reason': set([('fd', 'file'),
10794                                   ('file', 'f_flags'),
10795                                   ('file', 'f_mode'),
10796                                   ('file', 'f_op')])},
10797                   {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
10798                   {'call': 'inotify_rm_watch',
10799                    'reason': set([('fd', 'file')])},
10800                   {'call': 'socketpair',
10801                    'reason': set([('file', 'f_flags'),
10802                                   ('file', 'f_mode'),
10803                                   ('file', 'f_op')])},
10804                   {'call': 'utimensat', 'reason': set([('fd', 'file')])},
10805                   {'call': 'inotify_add_watch',
10806                    'reason': set([('fd', 'file')])},
10807                   {'call': 'preadv2', 'reason': set([('fd', 'file')])},
10808                   {'call': 'splice', 'reason': set([('fd', 'file')])},
10809                   {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
10810                   {'call': 'preadv', 'reason': set([('fd', 'file')])},
10811                   {'call': 'getpeername', 'reason': set([('fd', 'file')])},
10812                   {'call': 'shmat',
10813                    'reason': set([('file', 'f_flags'),
10814                                   ('file', 'f_mode'),
10815                                   ('file', 'f_op')])},
10816                   {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
10817                   {'call': 'socket',
```

```
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'pipe2',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
                {'call': 'ioctl', 'reason': set([('fd', 'file')])},
                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
                {'call': 'perf_event_open',
                 'reason': set([('fd', 'file'),
                                ('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'shmdt',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
                {'call': 'futimesat', 'reason': set([('fd', 'file')])},
                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
                {'call': 'acct',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'open',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
                {'call': 'mq_getsetattr',
                 'reason': set([('fd', 'file'),
                                ('file', 'f_flags'),
                                ('mq_attr', 'mq_curmsgs'),
                                ('mq_attr', 'mq_maxmsg'),
                                ('mq_attr', 'mq_msgsize'),
                                ('mqueue_inode_info', 'node_cache')])},
                {'call': 'dup',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
                {'call': 'setns',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
                {'call': 'listen', 'reason': set([('fd', 'file')])},
                {'call': 'copy_file_range',
                 'reason': set([('fd', 'file')])},
                {'call': 'mq_timedsend',
                 'reason': set([('fd', 'file'),
                                ('mq_attr', 'mq_curmsgs'),
                                ('mq_attr', 'mq_maxmsg'),
                                ('mq_attr', 'mq_msgsize'),
                                ('mqueue_inode_info', 'node_cache')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
                {'call': 'shmctl',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
                {'call': 'swapon',
                 'reason': set([('file', 'f_flags'),
```



```
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
                {'call': 'llseek', 'reason': set([('fd', 'file')])},
                {'call': 'mmap_pgoff',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
                {'call': 'readv', 'reason': set([('fd', 'file')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
                {'call': 'write', 'reason': set([('fd', 'file')])},
                {'call': 'mq_notify',
                 'reason': set([('fd', 'file'),
                                ('mq_attr', 'mq_curmsgs'),
                                ('mq_attr', 'mq_maxmsg'),
                                ('mq_attr', 'mq_msgsize'),
                                ('mqueue_inode_info', 'node_cache')])},
                {'call': 'sendfile', 'reason': set([('fd', 'file')])},
                {'call': 'mq_open',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op'),
                                ('mq_attr', 'mq_curmsgs'),
                                ('mq_attr', 'mq_maxmsg'),
                                ('mq_attr', 'mq_msgsize')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('file', 'f_op')])},
                {'call': 'bind', 'reason': set([('fd', 'file')])},
                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
 'mremap': [{'call': 'keyctl',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'msgrcv',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
            {'call': 'swapoff', 'reason': set([('mm_struct', 'map_count')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'ioprio_set',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'personality',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'remap_file_pages',
             'reason': set([('mm_struct', 'map_count'),
                            ('vm_area_struct', 'vm_end'),
                            ('vm_area_struct', 'vm_file'),
                            ('vm_area_struct', 'vm_flags'),
                            ('vm_area_struct', 'vm_next'),
                            ('vm_area_struct', 'vm_ops'),
                            ('vm_area_struct', 'vm_pgoff'),
                            ('vm_area_struct', 'vm_start')])},
            {'call': 'io_getevents',
             'reason': set([('mm_struct', 'map_count')])},
            {'call': 'getppid',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'personality')])},
```

```
10950             {'call': 'capget',
10951              'reason': set([('task_struct', 'personality')])},
10952             {'call': 'sched_setaffinity',
10953              'reason': set([('task_struct', 'personality')])},
10954             {'call': 'signal',
10955              'reason': set([('task_struct', 'personality')])},
10956             {'call': 'semtimedop',
10957              'reason': set([('task_struct', 'personality')])},
10958             {'call': 'umount',
10959              'reason': set([('task_struct', 'personality')])},
10960             {'call': 'sched_rr_get_interval',
10961              'reason': set([('task_struct', 'personality')])},
10962             {'call': 'rt_sigprocmask',
10963              'reason': set([('task_struct', 'personality')])},
10964             {'call': 'setsid',
10965              'reason': set([('task_struct', 'personality')])},
10966             {'call': 'sigaltstack',
10967              'reason': set([('task_struct', 'personality')])},
10968             {'call': 'sched_setattr',
10969              'reason': set([('task_struct', 'personality')])},
10970             {'call': 'migrate_pages',
10971              'reason': set([('mm_struct', 'map_count'),
10972                             ('task_struct', 'personality')])},
10973             {'call': 'getitimer',
10974              'reason': set([('task_struct', 'personality')])},
10975             {'call': 'setpgid',
10976              'reason': set([('task_struct', 'personality')])},
10977             {'call': 'getsid',
10978              'reason': set([('task_struct', 'personality')])},
10979             {'call': 'prlimit64',
10980              'reason': set([('task_struct', 'personality')])},
10981             {'call': 'perf_event_open',
10982              'reason': set([('task_struct', 'personality')])},
10983             {'call': 'shmdt',
10984              'reason': set([('mm_struct', 'map_count'),
10985                             ('vm_area_struct', 'vm_end'),
10986                             ('vm_area_struct', 'vm_file'),
10987                             ('vm_area_struct', 'vm_flags'),
10988                             ('vm_area_struct', 'vm_next'),
10989                             ('vm_area_struct', 'vm_ops'),
10990                             ('vm_area_struct', 'vm_pgoff'),
10991                             ('vm_area_struct', 'vm_start')])},
10992             {'call': 'rt_sigaction',
10993              'reason': set([('task_struct', 'personality')])},
10994             {'call': 'getpgid',
10995              'reason': set([('task_struct', 'personality')])},
10996             {'call': 'brk',
10997              'reason': set([('mm_struct', 'map_count'),
10998                             ('vm_area_struct', 'vm_end'),
10999                             ('vm_area_struct', 'vm_file'),
11000                             ('vm_area_struct', 'vm_flags'),
11001                             ('vm_area_struct', 'vm_next'),
11002                             ('vm_area_struct', 'vm_ops'),
11003                             ('vm_area_struct', 'vm_pgoff'),
11004                             ('vm_area_struct', 'vm_start')])},
11005             {'call': 'getpriority',
11006              'reason': set([('task_struct', 'personality')])},
11007             {'call': 'sigaction',
11008              'reason': set([('task_struct', 'personality')])},
11009             {'call': 'setns',
11010              'reason': set([('task_struct', 'personality')])},
11011             {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
11012             {'call': 'get_mempolicy',
11013              'reason': set([('mm_struct', 'map_count'),
11014                             ('vm_area_struct', 'vm_end'),
11015                             ('vm_area_struct', 'vm_file'),
```



```
11016                             ('vm_area_struct', 'vm_flags'),
11017                             ('vm_area_struct', 'vm_next'),
11018                             ('vm_area_struct', 'vm_ops'),
11019                             ('vm_area_struct', 'vm_pgoff'),
11020                             ('vm_area_struct', 'vm_start')])},
11021             {'call': 'get_robust_list',
11022              'reason': set([('task_struct', 'personality')])},
11023             {'call': 'mq_timedsend',
11024              'reason': set([('task_struct', 'personality')])},
11025             {'call': 'sched_getscheduler',
11026              'reason': set([('task_struct', 'personality')])},
11027             {'call': 'ptrace',
11028              'reason': set([('task_struct', 'personality')])},
11029             {'call': 'munlockall',
11030              'reason': set([('vm_area_struct', 'vm_end'),
11031                             ('vm_area_struct', 'vm_file'),
11032                             ('vm_area_struct', 'vm_flags'),
11033                             ('vm_area_struct', 'vm_next'),
11034                             ('vm_area_struct', 'vm_ops'),
11035                             ('vm_area_struct', 'vm_pgoff'),
11036                             ('vm_area_struct', 'vm_start')])},
11037             {'call': 'pkey_mprotect',
11038              'reason': set([('vm_area_struct', 'vm_end'),
11039                             ('vm_area_struct', 'vm_file'),
11040                             ('vm_area_struct', 'vm_flags'),
11041                             ('vm_area_struct', 'vm_next'),
11042                             ('vm_area_struct', 'vm_ops'),
11043                             ('vm_area_struct', 'vm_pgoff'),
11044                             ('vm_area_struct', 'vm_start')])},
11045             {'call': 'madvise',
11046              'reason': set([('vm_area_struct', 'vm_end'),
11047                             ('vm_area_struct', 'vm_file'),
11048                             ('vm_area_struct', 'vm_flags'),
11049                             ('vm_area_struct', 'vm_next'),
11050                             ('vm_area_struct', 'vm_ops'),
11051                             ('vm_area_struct', 'vm_pgoff'),
11052                             ('vm_area_struct', 'vm_start')])},
11053             {'call': 'sched_getattr',
11054              'reason': set([('task_struct', 'personality')])},
11055             {'call': 'getrusage',
11056              'reason': set([('mm_struct', 'map_count'),
11057                             ('task_struct', 'personality')])},
11058             {'call': 'sched_setscheduler',
11059              'reason': set([('task_struct', 'personality')])},
11060             {'call': 'setitimer',
11061              'reason': set([('task_struct', 'personality')])},
11062             {'call': 'ioprio_get',
11063              'reason': set([('task_struct', 'personality')])},
11064             {'call': 'vfork',
11065              'reason': set([('task_struct', 'personality')])},
11066             {'call': 'io_setup', 'reason': set([('mm_struct', 'map_count')])},
11067             {'call': 'mprotect',
11068              'reason': set([('vm_area_struct', 'vm_end'),
11069                             ('vm_area_struct', 'vm_file'),
11070                             ('vm_area_struct', 'vm_flags'),
11071                             ('vm_area_struct', 'vm_next'),
11072                             ('vm_area_struct', 'vm_ops'),
11073                             ('vm_area_struct', 'vm_pgoff'),
11074                             ('vm_area_struct', 'vm_start')])},
11075             {'call': 'mremap',
11076              'reason': set([('mm_struct', 'map_count'),
11077                             ('vm_area_struct', 'vm_end'),
11078                             ('vm_area_struct', 'vm_file'),
11079                             ('vm_area_struct', 'vm_flags'),
11080                             ('vm_area_struct', 'vm_next'),
11081                             ('vm_area_struct', 'vm_ops'),
```

```
                            ('vm_area_struct', 'vm_pgoff'),
                            ('vm_area_struct', 'vm_start')])},
            {'call': 'io_destroy',
             'reason': set([('mm_struct', 'map_count')])},
            {'call': 'mbind', 'reason': set([('mm_struct', 'map_count')])},
            {'call': 'prctl',
             'reason': set([('mm_struct', 'map_count'),
                            ('task_struct', 'personality'),
                            ('vm_area_struct', 'vm_end'),
                            ('vm_area_struct', 'vm_file'),
                            ('vm_area_struct', 'vm_flags'),
                            ('vm_area_struct', 'vm_next'),
                            ('vm_area_struct', 'vm_ops'),
                            ('vm_area_struct', 'vm_pgoff'),
                            ('vm_area_struct', 'vm_start')])},
            {'call': 'move_pages',
             'reason': set([('mm_struct', 'map_count'),
                            ('task_struct', 'personality')])},
            {'call': 'modify_ldt',
             'reason': set([('mm_struct', 'map_count')])},
            {'call': 'munlock',
             'reason': set([('vm_area_struct', 'vm_end'),
                            ('vm_area_struct', 'vm_file'),
                            ('vm_area_struct', 'vm_flags'),
                            ('vm_area_struct', 'vm_next'),
                            ('vm_area_struct', 'vm_ops'),
                            ('vm_area_struct', 'vm_pgoff'),
                            ('vm_area_struct', 'vm_start')])},
            {'call': 'setpriority',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'mincore',
             'reason': set([('mm_struct', 'map_count'),
                            ('vm_area_struct', 'vm_end'),
                            ('vm_area_struct', 'vm_file'),
                            ('vm_area_struct', 'vm_flags'),
                            ('vm_area_struct', 'vm_next'),
                            ('vm_area_struct', 'vm_ops'),
                            ('vm_area_struct', 'vm_pgoff'),
                            ('vm_area_struct', 'vm_start')])},
            {'call': 'clone',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_getparam',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'io_cancel',
             'reason': set([('mm_struct', 'map_count')])},
            {'call': 'mlockall',
             'reason': set([('vm_area_struct', 'vm_end'),
                            ('vm_area_struct', 'vm_file'),
                            ('vm_area_struct', 'vm_flags'),
                            ('vm_area_struct', 'vm_next'),
                            ('vm_area_struct', 'vm_ops'),
                            ('vm_area_struct', 'vm_pgoff'),
                            ('vm_area_struct', 'vm_start')])}],
 'msgctl': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'msgrcv',
             'reason': set([('ipc_namespace', 'msg_ctlmnb'),
                            ('mm_segment_t', 'seg')])},
            {'call': 'mq_unlink',
             'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
            {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'msgget',
             'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
            {'call': 'sched_getaffinity',
             'reason': set([('mm_segment_t', 'seg')])},
```



```
            {'call': 'sched_setparam',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'mq_timedreceive',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setaffinity',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'semtimedop',
             'reason': set([('ipc_namespace', 'msg_ctlmnb'),
                            ('mm_segment_t', 'seg')])},
            {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'semctl',
             'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
            {'call': 'shmget',
             'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'msgctl',
             'reason': set([('ipc_namespace', 'msg_ctlmnb'),
                            ('msqid64_ds', 'msg_qbytes')])},
            {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setattr',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'migrate_pages',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'semget',
             'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
            {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'shmat',
             'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
            {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'perf_event_open',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'rt_sigaction',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setns',
             'reason': set([('ipc_namespace', 'msg_ctlmnb'),
                            ('mm_segment_t', 'seg')])},
            {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'get_robust_list',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'mq_timedsend',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getscheduler',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'shmctl',
             'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
            {'call': 'sched_getattr',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setscheduler',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
```

```
11214             {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
11215             {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
11216             {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
11217             {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
11218             {'call': 'msgsnd',
11219              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
11220             {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
11221             {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
11222             {'call': 'mq_open',
11223              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
11224             {'call': 'sched_getparam',
11225              'reason': set([('mm_segment_t', 'seg')])}],
11226  'msgrcv': [{'call': 'msgrcv', 'reason': set([('msg_msg', 'm_ts')])},
11227             {'call': 'mq_timedreceive', 'reason': set([('msg_msg', 'm_ts')])},
11228             {'call': 'mq_timedsend', 'reason': set([('msg_msg', 'm_ts')])},
11229             {'call': 'msgsnd', 'reason': set([('msg_msg', 'm_ts')])}],
11230  'msgsnd': [{'call': 'msgrcv',
11231              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11232             {'call': 'mq_unlink',
11233              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11234             {'call': 'msgget',
11235              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11236             {'call': 'semtimedop',
11237              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11238             {'call': 'semctl',
11239              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11240             {'call': 'shmget',
11241              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11242             {'call': 'msgctl',
11243              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11244             {'call': 'semget',
11245              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11246             {'call': 'shmat',
11247              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11248             {'call': 'setns',
11249              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11250             {'call': 'shmctl',
11251              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11252             {'call': 'msgsnd',
11253              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
11254             {'call': 'mq_open',
11255              'reason': set([('ipc_namespace', 'msg_ctlmax')])}],
11256  'munlock': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
11257              {'call': 'rt_sigtimedwait',
11258               'reason': set([('task_struct', 'mm')])},
11259              {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
11260              {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
11261              {'call': 'sched_getaffinity',
11262               'reason': set([('task_struct', 'mm')])},
11263              {'call': 'sched_setparam',
11264               'reason': set([('task_struct', 'mm')])},
11265              {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
11266              {'call': 'remap_file_pages',
11267               'reason': set([('vm_area_struct', 'vm_end'),
11268                              ('vm_area_struct', 'vm_start')])},
11269              {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
11270              {'call': 'mq_timedreceive',
11271               'reason': set([('task_struct', 'mm')])},
11272              {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
11273              {'call': 'sched_setaffinity',
11274               'reason': set([('task_struct', 'mm')])},
11275              {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
11276              {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
11277              {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
11278              {'call': 'sched_rr_get_interval',
11279               'reason': set([('task_struct', 'mm')])},
```



```
11280             {'call': 'rt_sigprocmask',
11281              'reason': set([('task_struct', 'mm')])},
11282             {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
11283             {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
11284             {'call': 'sched_setattr',
11285              'reason': set([('task_struct', 'mm')])},
11286             {'call': 'migrate_pages',
11287              'reason': set([('task_struct', 'mm')])},
11288             {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
11289             {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
11290             {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
11291             {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
11292             {'call': 'perf_event_open',
11293              'reason': set([('task_struct', 'mm')])},
11294             {'call': 'shmdt',
11295              'reason': set([('vm_area_struct', 'vm_end'),
11296                             ('vm_area_struct', 'vm_start')])},
11297             {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
11298             {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
11299             {'call': 'brk',
11300              'reason': set([('vm_area_struct', 'vm_end'),
11301                             ('vm_area_struct', 'vm_start')])},
11302             {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
11303             {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
11304             {'call': 'setns', 'reason': set([('task_struct', 'mm')])},
11305             {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
11306             {'call': 'get_mempolicy',
11307              'reason': set([('vm_area_struct', 'vm_end'),
11308                             ('vm_area_struct', 'vm_start')])},
11309             {'call': 'get_robust_list',
11310              'reason': set([('task_struct', 'mm')])},
11311             {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
11312             {'call': 'sched_getscheduler',
11313              'reason': set([('task_struct', 'mm')])},
11314             {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
11315             {'call': 'munlockall',
11316              'reason': set([('vm_area_struct', 'vm_end'),
11317                             ('vm_area_struct', 'vm_start')])},
11318             {'call': 'pkey_mprotect',
11319              'reason': set([('vm_area_struct', 'vm_end'),
11320                             ('vm_area_struct', 'vm_start')])},
11321             {'call': 'madvise',
11322              'reason': set([('vm_area_struct', 'vm_end'),
11323                             ('vm_area_struct', 'vm_start')])},
11324             {'call': 'sched_getattr',
11325              'reason': set([('task_struct', 'mm')])},
11326             {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
11327             {'call': 'sched_setscheduler',
11328              'reason': set([('task_struct', 'mm')])},
11329             {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
11330             {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
11331             {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
11332             {'call': 'mprotect',
11333              'reason': set([('vm_area_struct', 'vm_end'),
11334                             ('vm_area_struct', 'vm_start')])},
11335             {'call': 'mremap',
11336              'reason': set([('vm_area_struct', 'vm_end'),
11337                             ('vm_area_struct', 'vm_start')])},
11338             {'call': 'prctl',
11339              'reason': set([('task_struct', 'mm'),
11340                             ('vm_area_struct', 'vm_end'),
11341                             ('vm_area_struct', 'vm_start')])},
11342             {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
11343             {'call': 'munlock',
11344              'reason': set([('vm_area_struct', 'vm_end'),
11345                             ('vm_area_struct', 'vm_start')])},
```

```
11346             {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
11347             {'call': 'mincore',
11348              'reason': set([('vm_area_struct', 'vm_end'),
11349                             ('vm_area_struct', 'vm_start')])},
11350             {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
11351             {'call': 'sched_getparam',
11352              'reason': set([('task_struct', 'mm')])},
11353             {'call': 'mlockall',
11354              'reason': set([('vm_area_struct', 'vm_end'),
11355                             ('vm_area_struct', 'vm_start')])}],
11356  'munlockall': [{'call': 'remap_file_pages',
11357                  'reason': set([('vm_area_struct', 'vm_end'),
11358                                 ('vm_area_struct', 'vm_start')])},
11359                 {'call': 'shmdt',
11360                  'reason': set([('vm_area_struct', 'vm_end'),
11361                                 ('vm_area_struct', 'vm_start')])},
11362                 {'call': 'brk',
11363                  'reason': set([('vm_area_struct', 'vm_end'),
11364                                 ('vm_area_struct', 'vm_start')])},
11365                 {'call': 'get_mempolicy',
11366                  'reason': set([('vm_area_struct', 'vm_end'),
11367                                 ('vm_area_struct', 'vm_start')])},
11368                 {'call': 'munlockall',
11369                  'reason': set([('vm_area_struct', 'vm_end'),
11370                                 ('vm_area_struct', 'vm_start')])},
11371                 {'call': 'pkey_mprotect',
11372                  'reason': set([('vm_area_struct', 'vm_end'),
11373                                 ('vm_area_struct', 'vm_start')])},
11374                 {'call': 'madvise',
11375                  'reason': set([('vm_area_struct', 'vm_end'),
11376                                 ('vm_area_struct', 'vm_start')])},
11377                 {'call': 'mprotect',
11378                  'reason': set([('vm_area_struct', 'vm_end'),
11379                                 ('vm_area_struct', 'vm_start')])},
11380                 {'call': 'mremap',
11381                  'reason': set([('vm_area_struct', 'vm_end'),
11382                                 ('vm_area_struct', 'vm_start')])},
11383                 {'call': 'prctl',
11384                  'reason': set([('vm_area_struct', 'vm_end'),
11385                                 ('vm_area_struct', 'vm_start')])},
11386                 {'call': 'munlock',
11387                  'reason': set([('vm_area_struct', 'vm_end'),
11388                                 ('vm_area_struct', 'vm_start')])},
11389                 {'call': 'mincore',
11390                  'reason': set([('vm_area_struct', 'vm_end'),
11391                                 ('vm_area_struct', 'vm_start')])},
11392                 {'call': 'mlockall',
11393                  'reason': set([('vm_area_struct', 'vm_end'),
11394                                 ('vm_area_struct', 'vm_start')])}],
11395  'name_to_handle_at': [{'call': 'name_to_handle_at',
11396                         'reason': set([('file_handle', 'handle_bytes')])},
11397                        {'call': 'syncfs',
11398                         'reason': set([('super_block', 's_export_op')])},
11399                        {'call': 'eventfd2',
11400                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11401                        {'call': 'swapoff',
11402                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11403                        {'call': 'pivot_root',
11404                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11405                        {'call': 'memfd_create',
11406                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11407                        {'call': 'remap_file_pages',
11408                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11409                        {'call': 'dup3',
11410                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11411                        {'call': 'ustat',
```



```
11412                         'reason': set([('super_block', 's_export_op')])},
11413                        {'call': 'unshare',
11414                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11415                        {'call': 'umount',
11416                         'reason': set([('super_block', 's_export_op')])},
11417                        {'call': 'epoll_create1',
11418                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11419                        {'call': 'epoll_ctl',
11420                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11421                        {'call': 'flock',
11422                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11423                        {'call': 'openat',
11424                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11425                        {'call': 'lookup_dcookie',
11426                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11427                        {'call': 'uselib',
11428                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11429                        {'call': 'accept4',
11430                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11431                        {'call': 'socketpair',
11432                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11433                        {'call': 'getcwd',
11434                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11435                        {'call': 'shmat',
11436                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11437                        {'call': 'socket',
11438                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11439                        {'call': 'pipe2',
11440                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11441                        {'call': 'perf_event_open',
11442                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11443                        {'call': 'shmdt',
11444                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11445                        {'call': 'quotactl',
11446                         'reason': set([('path', 'dentry'),
11447                                        ('path', 'mnt'),
11448                                        ('super_block', 's_export_op')])},
11449                        {'call': 'acct',
11450                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11451                        {'call': 'open',
11452                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11453                        {'call': 'dup',
11454                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11455                        {'call': 'setns',
11456                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11457                        {'call': 'shmctl',
11458                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11459                        {'call': 'swapon',
11460                         'reason': set([('path', 'dentry'),
11461                                        ('path', 'mnt'),
11462                                        ('super_block', 's_export_op')])},
11463                        {'call': 'mmap_pgoff',
11464                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11465                        {'call': 'mq_open',
11466                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
11467                        {'call': 'open_by_handle_at',
11468                         'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
11469  'nanosleep': [{'call': 'rt_sigtimedwait',
11470                 'reason': set([('timespec', 'tv_nsec'),
11471                                ('timespec', 'tv_sec')])},
11472                {'call': 'mq_unlink',
11473                 'reason': set([('timespec', 'tv_nsec'),
11474                                ('timespec', 'tv_sec')])},
11475                {'call': 'swapoff',
11476                 'reason': set([('timespec', 'tv_nsec'),
11477                                ('timespec', 'tv_sec')])},
```

```
11478                {'call': 'fchmod',
11479                 'reason': set([('timespec', 'tv_nsec'),
11480                                ('timespec', 'tv_sec')])},
11481                {'call': 'memfd_create',
11482                 'reason': set([('timespec', 'tv_nsec'),
11483                                ('timespec', 'tv_sec')])},
11484                {'call': 'readlinkat',
11485                 'reason': set([('timespec', 'tv_nsec'),
11486                                ('timespec', 'tv_sec')])},
11487                {'call': 'io_getevents',
11488                 'reason': set([('timespec', 'tv_nsec'),
11489                                ('timespec', 'tv_sec')])},
11490                {'call': 'fchown',
11491                 'reason': set([('timespec', 'tv_nsec'),
11492                                ('timespec', 'tv_sec')])},
11493                {'call': 'mq_timedreceive',
11494                 'reason': set([('timespec', 'tv_nsec'),
11495                                ('timespec', 'tv_sec')])},
11496                {'call': 'utime',
11497                 'reason': set([('timespec', 'tv_nsec'),
11498                                ('timespec', 'tv_sec')])},
11499                {'call': 'semtimedop',
11500                 'reason': set([('timespec', 'tv_nsec'),
11501                                ('timespec', 'tv_sec')])},
11502                {'call': 'settimeofday',
11503                 'reason': set([('timespec', 'tv_nsec'),
11504                                ('timespec', 'tv_sec')])},
11505                {'call': 'sched_rr_get_interval',
11506                 'reason': set([('timespec', 'tv_nsec'),
11507                                ('timespec', 'tv_sec')])},
11508                {'call': 'timerfd_gettime',
11509                 'reason': set([('timespec', 'tv_nsec'),
11510                                ('timespec', 'tv_sec')])},
11511                {'call': 'pselect6',
11512                 'reason': set([('timespec', 'tv_nsec'),
11513                                ('timespec', 'tv_sec')])},
11514                {'call': 'uselib',
11515                 'reason': set([('timespec', 'tv_nsec'),
11516                                ('timespec', 'tv_sec')])},
11517                {'call': 'fchmodat',
11518                 'reason': set([('timespec', 'tv_nsec'),
11519                                ('timespec', 'tv_sec')])},
11520                {'call': 'inotify_add_watch',
11521                 'reason': set([('timespec', 'tv_nsec'),
11522                                ('timespec', 'tv_sec')])},
11523                {'call': 'timer_settime',
11524                 'reason': set([('timespec', 'tv_nsec'),
11525                                ('timespec', 'tv_sec')])},
11526                {'call': 'ftruncate',
11527                 'reason': set([('timespec', 'tv_nsec'),
11528                                ('timespec', 'tv_sec')])},
11529                {'call': 'timer_gettime',
11530                 'reason': set([('timespec', 'tv_nsec'),
11531                                ('timespec', 'tv_sec')])},
11532                {'call': 'ioctl',
11533                 'reason': set([('timespec', 'tv_nsec'),
11534                                ('timespec', 'tv_sec')])},
11535                {'call': 'linkat',
11536                 'reason': set([('timespec', 'tv_nsec'),
11537                                ('timespec', 'tv_sec')])},
11538                {'call': 'stime',
11539                 'reason': set([('timespec', 'tv_nsec'),
11540                                ('timespec', 'tv_sec')])},
11541                {'call': 'futimesat',
11542                 'reason': set([('timespec', 'tv_nsec'),
11543                                ('timespec', 'tv_sec')])},
```



```
11544                {'call': 'poll',
11545                 'reason': set([('timespec', 'tv_nsec'),
11546                                ('timespec', 'tv_sec')])},
11547                {'call': 'select',
11548                 'reason': set([('timespec', 'tv_nsec'),
11549                                ('timespec', 'tv_sec')])},
11550                {'call': 'unlink',
11551                 'reason': set([('timespec', 'tv_nsec'),
11552                                ('timespec', 'tv_sec')])},
11553                {'call': 'nanosleep',
11554                 'reason': set([('timespec', 'tv_nsec'),
11555                                ('timespec', 'tv_sec')])},
11556                {'call': 'mq_getsetattr',
11557                 'reason': set([('timespec', 'tv_nsec'),
11558                                ('timespec', 'tv_sec')])},
11559                {'call': 'faccessat',
11560                 'reason': set([('timespec', 'tv_nsec'),
11561                                ('timespec', 'tv_sec')])},
11562                {'call': 'mq_timedsend',
11563                 'reason': set([('timespec', 'tv_nsec'),
11564                                ('timespec', 'tv_sec')])},
11565                {'call': 'swapon',
11566                 'reason': set([('timespec', 'tv_nsec'),
11567                                ('timespec', 'tv_sec')])},
11568                {'call': 'epoll_wait',
11569                 'reason': set([('timespec', 'tv_nsec'),
11570                                ('timespec', 'tv_sec')])},
11571                {'call': 'fchownat',
11572                 'reason': set([('timespec', 'tv_nsec'),
11573                                ('timespec', 'tv_sec')])},
11574                {'call': 'fstat',
11575                 'reason': set([('timespec', 'tv_nsec'),
11576                                ('timespec', 'tv_sec')])},
11577                {'call': 'timerfd_settime',
11578                 'reason': set([('timespec', 'tv_nsec'),
11579                                ('timespec', 'tv_sec')])},
11580                {'call': 'mq_notify',
11581                 'reason': set([('timespec', 'tv_nsec'),
11582                                ('timespec', 'tv_sec')])},
11583                {'call': 'sendfile',
11584                 'reason': set([('timespec', 'tv_nsec'),
11585                                ('timespec', 'tv_sec')])},
11586                {'call': 'newfstat',
11587                 'reason': set([('timespec', 'tv_nsec'),
11588                                ('timespec', 'tv_sec')])},
11589                {'call': 'clock_nanosleep',
11590                 'reason': set([('timespec', 'tv_nsec'),
11591                                ('timespec', 'tv_sec')])},
11592                {'call': 'unlinkat',
11593                 'reason': set([('timespec', 'tv_nsec'),
11594                                ('timespec', 'tv_sec')])},
11595                {'call': 'futex',
11596                 'reason': set([('timespec', 'tv_nsec'),
11597                                ('timespec', 'tv_sec')])},
11598                {'call': 'recvmmsg',
11599                 'reason': set([('timespec', 'tv_nsec'),
11600                                ('timespec', 'tv_sec')])},
11601                {'call': 'sendfile64',
11602                 'reason': set([('timespec', 'tv_nsec'),
11603                                ('timespec', 'tv_sec')])},
11604                {'call': 'ppoll',
11605                 'reason': set([('timespec', 'tv_nsec'),
11606                                ('timespec', 'tv_sec')])}],
11607  'newfstat': [{'call': 'newlstat',
11608                'reason': set([('compat_stat', 'st_ino'),
11609                               ('compat_stat', 'st_nlink'),
```

```
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])},
                {'call': 'newfstatat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])},
                {'call': 'newstat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])},
                {'call': 'fstat',
                 'reason': set([('kstat', 'dev'),
                                ('kstat', 'ino'),
                                ('kstat', 'nlink'),
                                ('kstat', 'rdev'),
                                ('kstat', 'size')])},
                {'call': 'newfstat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('kstat', 'dev'),
                                ('kstat', 'ino'),
                                ('kstat', 'nlink'),
                                ('kstat', 'rdev'),
                                ('kstat', 'size'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])}],
 'newfstatat': [{'call': 'newlstat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])},
                {'call': 'newfstatat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])},
                {'call': 'newstat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])},
                {'call': 'fstat',
                 'reason': set([('kstat', 'dev'),
                                ('kstat', 'ino'),
                                ('kstat', 'nlink'),
                                ('kstat', 'rdev'),
                                ('kstat', 'size')])},
                {'call': 'newfstat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('kstat', 'dev'),
                                ('kstat', 'ino'),
                                ('kstat', 'nlink'),
                                ('kstat', 'rdev'),
                                ('kstat', 'size'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])}],
 'newlstat': [{'call': 'newlstat',
               'reason': set([('compat_stat', 'st_ino'),
                              ('compat_stat', 'st_nlink'),
                              ('stat', 'st_ino'),
                              ('stat', 'st_nlink')])},
              {'call': 'newfstatat',
               'reason': set([('compat_stat', 'st_ino'),
```



```
                              ('compat_stat', 'st_nlink'),
                              ('stat', 'st_ino'),
                              ('stat', 'st_nlink')])},
              {'call': 'newstat',
               'reason': set([('compat_stat', 'st_ino'),
                              ('compat_stat', 'st_nlink'),
                              ('stat', 'st_ino'),
                              ('stat', 'st_nlink')])},
              {'call': 'fstat',
               'reason': set([('kstat', 'dev'),
                              ('kstat', 'ino'),
                              ('kstat', 'nlink'),
                              ('kstat', 'rdev'),
                              ('kstat', 'size')])},
              {'call': 'newfstat',
               'reason': set([('compat_stat', 'st_ino'),
                              ('compat_stat', 'st_nlink'),
                              ('kstat', 'dev'),
                              ('kstat', 'ino'),
                              ('kstat', 'nlink'),
                              ('kstat', 'rdev'),
                              ('kstat', 'size'),
                              ('stat', 'st_ino'),
                              ('stat', 'st_nlink')])}],
 'newstat': [{'call': 'newlstat',
              'reason': set([('compat_stat', 'st_ino'),
                             ('compat_stat', 'st_nlink'),
                             ('stat', 'st_ino'),
                             ('stat', 'st_nlink')])},
             {'call': 'newfstatat',
              'reason': set([('compat_stat', 'st_ino'),
                             ('compat_stat', 'st_nlink'),
                             ('stat', 'st_ino'),
                             ('stat', 'st_nlink')])},
             {'call': 'newstat',
              'reason': set([('compat_stat', 'st_ino'),
                             ('compat_stat', 'st_nlink'),
                             ('stat', 'st_ino'),
                             ('stat', 'st_nlink')])},
             {'call': 'fstat',
              'reason': set([('kstat', 'dev'),
                             ('kstat', 'ino'),
                             ('kstat', 'nlink'),
                             ('kstat', 'rdev'),
                             ('kstat', 'size')])},
             {'call': 'newfstat',
              'reason': set([('compat_stat', 'st_ino'),
                             ('compat_stat', 'st_nlink'),
                             ('kstat', 'dev'),
                             ('kstat', 'ino'),
                             ('kstat', 'nlink'),
                             ('kstat', 'rdev'),
                             ('kstat', 'size'),
                             ('stat', 'st_ino'),
                             ('stat', 'st_nlink')])}],
 'newuname': [{'call': 'keyctl',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'msgrcv',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'kill',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sched_getaffinity',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sched_setparam',
```

```
11742                'reason': set([('task_struct', 'personality')])},
11743               {'call': 'ioprio_set',
11744                'reason': set([('task_struct', 'personality')])},
11745               {'call': 'personality',
11746                'reason': set([('task_struct', 'personality')])},
11747               {'call': 'getppid',
11748                'reason': set([('task_struct', 'personality')])},
11749               {'call': 'mq_timedreceive',
11750                'reason': set([('task_struct', 'personality')])},
11751               {'call': 'capget',
11752                'reason': set([('task_struct', 'personality')])},
11753               {'call': 'sched_setaffinity',
11754                'reason': set([('task_struct', 'personality')])},
11755               {'call': 'signal',
11756                'reason': set([('task_struct', 'personality')])},
11757               {'call': 'semtimedop',
11758                'reason': set([('task_struct', 'personality')])},
11759               {'call': 'umount',
11760                'reason': set([('task_struct', 'personality')])},
11761               {'call': 'sched_rr_get_interval',
11762                'reason': set([('task_struct', 'personality')])},
11763               {'call': 'rt_sigprocmask',
11764                'reason': set([('task_struct', 'personality')])},
11765               {'call': 'setsid',
11766                'reason': set([('task_struct', 'personality')])},
11767               {'call': 'sigaltstack',
11768                'reason': set([('task_struct', 'personality')])},
11769               {'call': 'sched_setattr',
11770                'reason': set([('task_struct', 'personality')])},
11771               {'call': 'migrate_pages',
11772                'reason': set([('task_struct', 'personality')])},
11773               {'call': 'getitimer',
11774                'reason': set([('task_struct', 'personality')])},
11775               {'call': 'setpgid',
11776                'reason': set([('task_struct', 'personality')])},
11777               {'call': 'getsid',
11778                'reason': set([('task_struct', 'personality')])},
11779               {'call': 'prlimit64',
11780                'reason': set([('task_struct', 'personality')])},
11781               {'call': 'perf_event_open',
11782                'reason': set([('task_struct', 'personality')])},
11783               {'call': 'rt_sigaction',
11784                'reason': set([('task_struct', 'personality')])},
11785               {'call': 'getpgid',
11786                'reason': set([('task_struct', 'personality')])},
11787               {'call': 'getpriority',
11788                'reason': set([('task_struct', 'personality')])},
11789               {'call': 'sigaction',
11790                'reason': set([('task_struct', 'personality')])},
11791               {'call': 'setns',
11792                'reason': set([('task_struct', 'personality')])},
11793               {'call': 'fork',
11794                'reason': set([('task_struct', 'personality')])},
11795               {'call': 'get_robust_list',
11796                'reason': set([('task_struct', 'personality')])},
11797               {'call': 'mq_timedsend',
11798                'reason': set([('task_struct', 'personality')])},
11799               {'call': 'sched_getscheduler',
11800                'reason': set([('task_struct', 'personality')])},
11801               {'call': 'ptrace',
11802                'reason': set([('task_struct', 'personality')])},
11803               {'call': 'sched_getattr',
11804                'reason': set([('task_struct', 'personality')])},
11805               {'call': 'getrusage',
11806                'reason': set([('task_struct', 'personality')])},
11807               {'call': 'sched_setscheduler',
```



```
11808                'reason': set([('task_struct', 'personality')])},
11809               {'call': 'setitimer',
11810                'reason': set([('task_struct', 'personality')])},
11811               {'call': 'ioprio_get',
11812                'reason': set([('task_struct', 'personality')])},
11813               {'call': 'vfork',
11814                'reason': set([('task_struct', 'personality')])},
11815               {'call': 'prctl',
11816                'reason': set([('task_struct', 'personality')])},
11817               {'call': 'move_pages',
11818                'reason': set([('task_struct', 'personality')])},
11819               {'call': 'setpriority',
11820                'reason': set([('task_struct', 'personality')])},
11821               {'call': 'clone',
11822                'reason': set([('task_struct', 'personality')])},
11823               {'call': 'sched_getparam',
11824                'reason': set([('task_struct', 'personality')])}],
11825  'old_getrlimit': [{'call': 'setrlimit',
11826                     'reason': set([('rlimit', 'rlim_cur'),
11827                                    ('rlimit', 'rlim_max')])},
11828                    {'call': 'old_getrlimit',
11829                     'reason': set([('rlimit', 'rlim_cur'),
11830                                    ('rlimit', 'rlim_max')])},
11831                    {'call': 'prlimit64',
11832                     'reason': set([('rlimit', 'rlim_cur'),
11833                                    ('rlimit', 'rlim_max')])}],
11834  'old_readdir': [{'call': 'syncfs',
11835                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11836                  {'call': 'vmsplice',
11837                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11838                  {'call': 'pwritev64',
11839                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11840                  {'call': 'fremovexattr',
11841                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11842                  {'call': 'readahead',
11843                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11844                  {'call': 'getdents',
11845                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11846                  {'call': 'writev',
11847                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11848                  {'call': 'preadv64',
11849                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11850                  {'call': 'fchmod',
11851                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11852                  {'call': 'pread64',
11853                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11854                  {'call': 'signalfd4',
11855                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11856                  {'call': 'read',
11857                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11858                  {'call': 'fchown',
11859                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11860                  {'call': 'mq_timedreceive',
11861                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11862                  {'call': 'utime',
11863                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11864                  {'call': 'fsync',
11865                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11866                  {'call': 'bpf',
11867                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11868                  {'call': 'recvfrom',
11869                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11870                  {'call': 'fsetxattr',
11871                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
11872                  {'call': 'sendto',
11873                   'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
                {'call': 'tee',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sync_file_range',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'lseek',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'connect',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'getsockname',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'epoll_ctl',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'flock',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'pwritev',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fchdir',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'accept4',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'old_readdir',
                 'reason': set([('compat_readdir_callback', 'result'),
                                ('fd', 'file'),
                                ('fd', 'flags'),
                                ('readdir_callback', 'result')])},
                {'call': 'inotify_rm_watch',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'utimensat',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'preadv2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'splice',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'ftruncate',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'preadv',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'getpeername',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'setsockopt',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fcntl',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'ioctl',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'pwrite64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'perf_event_open',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'pwritev64v2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'futimesat',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'pwritev2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'shutdown',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'getsockopt',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mq_getsetattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fdatasync',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'getdents64',
```



```
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'listen',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'copy_file_range',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mq_timedsend',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fgetxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fcntl64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fallocate',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'epoll_wait',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'lseek',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'preadv64v2',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'readv',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fstatfs',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'fstatfs64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'write',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'mq_notify',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sendfile',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'bind',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'flistxattr',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
                {'call': 'sendfile64',
                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
 'olduname': [{'call': 'keyctl',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'msgrcv',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'kill',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'sched_getaffinity',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'sched_setparam',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'ioprio_set',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'personality',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'getppid',
               'reason': set([('mm_segment_t', 'seg'),
                              ('task_struct', 'personality')])},
              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_timedreceive',
               'reason': set([('mm_segment_t', 'seg'),
```

```
12006                                 ('task_struct', 'personality')])},
12007                {'call': 'capget',
12008                 'reason': set([('mm_segment_t', 'seg'),
12009                                 ('task_struct', 'personality')])},
12010                {'call': 'sched_setaffinity',
12011                 'reason': set([('mm_segment_t', 'seg'),
12012                                 ('task_struct', 'personality')])},
12013                {'call': 'signal',
12014                 'reason': set([('mm_segment_t', 'seg'),
12015                                 ('task_struct', 'personality')])},
12016                {'call': 'semtimedop',
12017                 'reason': set([('mm_segment_t', 'seg'),
12018                                 ('task_struct', 'personality')])},
12019                {'call': 'umount',
12020                 'reason': set([('mm_segment_t', 'seg'),
12021                                 ('task_struct', 'personality')])},
12022                {'call': 'sched_rr_get_interval',
12023                 'reason': set([('mm_segment_t', 'seg'),
12024                                 ('task_struct', 'personality')])},
12025                {'call': 'rt_sigprocmask',
12026                 'reason': set([('mm_segment_t', 'seg'),
12027                                 ('task_struct', 'personality')])},
12028                {'call': 'setsid',
12029                 'reason': set([('mm_segment_t', 'seg'),
12030                                 ('task_struct', 'personality')])},
12031                {'call': 'sigaltstack',
12032                 'reason': set([('mm_segment_t', 'seg'),
12033                                 ('task_struct', 'personality')])},
12034                {'call': 'sched_setattr',
12035                 'reason': set([('mm_segment_t', 'seg'),
12036                                 ('task_struct', 'personality')])},
12037                {'call': 'migrate_pages',
12038                 'reason': set([('mm_segment_t', 'seg'),
12039                                 ('task_struct', 'personality')])},
12040                {'call': 'getitimer',
12041                 'reason': set([('mm_segment_t', 'seg'),
12042                                 ('task_struct', 'personality')])},
12043                {'call': 'setpgid',
12044                 'reason': set([('mm_segment_t', 'seg'),
12045                                 ('task_struct', 'personality')])},
12046                {'call': 'getsid',
12047                 'reason': set([('mm_segment_t', 'seg'),
12048                                 ('task_struct', 'personality')])},
12049                {'call': 'prlimit64',
12050                 'reason': set([('mm_segment_t', 'seg'),
12051                                 ('task_struct', 'personality')])},
12052                {'call': 'perf_event_open',
12053                 'reason': set([('mm_segment_t', 'seg'),
12054                                 ('task_struct', 'personality')])},
12055                {'call': 'rt_sigaction',
12056                 'reason': set([('mm_segment_t', 'seg'),
12057                                 ('task_struct', 'personality')])},
12058                {'call': 'getpgid',
12059                 'reason': set([('mm_segment_t', 'seg'),
12060                                 ('task_struct', 'personality')])},
12061                {'call': 'getpriority',
12062                 'reason': set([('mm_segment_t', 'seg'),
12063                                 ('task_struct', 'personality')])},
12064                {'call': 'sigaction',
12065                 'reason': set([('mm_segment_t', 'seg'),
12066                                 ('task_struct', 'personality')])},
12067                {'call': 'setns',
12068                 'reason': set([('mm_segment_t', 'seg'),
12069                                 ('task_struct', 'personality')])},
12070                {'call': 'fork',
12071                 'reason': set([('mm_segment_t', 'seg'),
```



```
12072                                 ('task_struct', 'personality')])},
12073                {'call': 'get_robust_list',
12074                 'reason': set([('mm_segment_t', 'seg'),
12075                                 ('task_struct', 'personality')])},
12076                {'call': 'mq_timedsend',
12077                 'reason': set([('mm_segment_t', 'seg'),
12078                                 ('task_struct', 'personality')])},
12079                {'call': 'sched_getscheduler',
12080                 'reason': set([('mm_segment_t', 'seg'),
12081                                 ('task_struct', 'personality')])},
12082                {'call': 'ptrace',
12083                 'reason': set([('mm_segment_t', 'seg'),
12084                                 ('task_struct', 'personality')])},
12085                {'call': 'sched_getattr',
12086                 'reason': set([('mm_segment_t', 'seg'),
12087                                 ('task_struct', 'personality')])},
12088                {'call': 'getrusage',
12089                 'reason': set([('mm_segment_t', 'seg'),
12090                                 ('task_struct', 'personality')])},
12091                {'call': 'sched_setscheduler',
12092                 'reason': set([('mm_segment_t', 'seg'),
12093                                 ('task_struct', 'personality')])},
12094                {'call': 'setitimer',
12095                 'reason': set([('mm_segment_t', 'seg'),
12096                                 ('task_struct', 'personality')])},
12097                {'call': 'ioprio_get',
12098                 'reason': set([('mm_segment_t', 'seg'),
12099                                 ('task_struct', 'personality')])},
12100                {'call': 'vfork',
12101                 'reason': set([('mm_segment_t', 'seg'),
12102                                 ('task_struct', 'personality')])},
12103                {'call': 'prctl',
12104                 'reason': set([('mm_segment_t', 'seg'),
12105                                 ('task_struct', 'personality')])},
12106                {'call': 'move_pages',
12107                 'reason': set([('mm_segment_t', 'seg'),
12108                                 ('task_struct', 'personality')])},
12109                {'call': 'setpriority',
12110                 'reason': set([('mm_segment_t', 'seg'),
12111                                 ('task_struct', 'personality')])},
12112                {'call': 'clone',
12113                 'reason': set([('mm_segment_t', 'seg'),
12114                                 ('task_struct', 'personality')])},
12115                {'call': 'sched_getparam',
12116                 'reason': set([('mm_segment_t', 'seg'),
12117                                 ('task_struct', 'personality')])}],
12118  'open_by_handle_at': [{'call': 'eventfd2',
12119                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12120                        {'call': 'swapoff',
12121                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12122                        {'call': 'pivot_root',
12123                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12124                        {'call': 'memfd_create',
12125                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12126                        {'call': 'remap_file_pages',
12127                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12128                        {'call': 'dup3',
12129                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12130                        {'call': 'unshare',
12131                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12132                        {'call': 'epoll_create1',
12133                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12134                        {'call': 'epoll_ctl',
12135                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12136                        {'call': 'flock',
12137                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```

```
                        {'call': 'openat',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'lookup_dcookie',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'uselib',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'accept4',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'socketpair',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'getcwd',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'shmat',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'socket',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'pipe2',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'perf_event_open',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'shmdt',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'quotactl',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'acct',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'open',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'dup',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'setns',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'shmctl',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'swapon',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'mmap_pgoff',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'mq_open',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                        {'call': 'open_by_handle_at',
                         'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
    'perf_event_open': [{'call': 'syncfs',
                         'reason': set([('fd', 'file'),
                                        ('fd', 'flags'),
                                        ('list_head', 'prev')])},
                        {'call': 'keyctl',
                         'reason': set([('list_head', 'prev'),
                                        ('mm_segment_t', 'seg'),
                                        ('task_struct', 'flags')])},
                        {'call': 'rt_sigtimedwait',
                         'reason': set([('list_head', 'prev'),
                                        ('mm_segment_t', 'seg'),
                                        ('task_struct', 'flags')])},
                        {'call': 'vmsplice',
                         'reason': set([('fd', 'file'),
                                        ('fd', 'flags'),
                                        ('list_head', 'prev')])},
                        {'call': 'msgrcv',
                         'reason': set([('list_head', 'prev'),
                                        ('mm_segment_t', 'seg'),
                                        ('task_struct', 'flags')])},
                        {'call': 'eventfd2',
                         'reason': set([('file', 'f_op'),
                                        ('list_head', 'prev')])},
                        {'call': 'mq_unlink',
```



```
                         'reason': set([('list_head', 'prev')])},
                        {'call': 'pwritev64',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'kill',
                         'reason': set([('list_head', 'prev'),
                                        ('mm_segment_t', 'seg'),
                                        ('task_struct', 'flags')])},
                        {'call': 'swapoff',
                         'reason': set([('file', 'f_op'),
                                        ('list_head', 'prev')])},
                        {'call': 'fremovexattr',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'readahead',
                         'reason': set([('fd', 'file'),
                                        ('fd', 'flags'),
                                        ('list_head', 'prev')])},
                        {'call': 'getdents',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'timer_delete',
                         'reason': set([('list_head', 'prev')])},
                        {'call': 'sched_getaffinity',
                         'reason': set([('list_head', 'prev'),
                                        ('mm_segment_t', 'seg'),
                                        ('task_struct', 'flags')])},
                        {'call': 'writev',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'preadv64',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'sched_setparam',
                         'reason': set([('list_head', 'prev'),
                                        ('mm_segment_t', 'seg'),
                                        ('task_struct', 'flags')])},
                        {'call': 'fchmod',
                         'reason': set([('fd', 'file'),
                                        ('fd', 'flags'),
                                        ('list_head', 'prev')])},
                        {'call': 'setgid',
                         'reason': set([('list_head', 'prev')])},
                        {'call': 'pread64',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'pivot_root',
                         'reason': set([('list_head', 'prev')])},
                        {'call': 'signalfd4',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'memfd_create',
                         'reason': set([('file', 'f_op'),
                                        ('list_head', 'prev')])},
                        {'call': 'ioprio_set',
                         'reason': set([('list_head', 'prev'),
                                        ('mm_segment_t', 'seg'),
                                        ('task_struct', 'flags')])},
                        {'call': 'delete_module',
                         'reason': set([('list_head', 'prev')])},
                        {'call': 'remap_file_pages',
                         'reason': set([('file', 'f_op'),
                                        ('list_head', 'prev')])},
                        {'call': 'dup3',
                         'reason': set([('file', 'f_op'),
                                        ('list_head', 'prev')])},
                        {'call': 'readlinkat',
                         'reason': set([('list_head', 'prev')])},
                        {'call': 'read',
                         'reason': set([('fd', 'file'), ('fd', 'flags')])},
                        {'call': 'io_getevents',
                         'reason': set([('list_head', 'prev')])},
                        {'call': 'getppid',
```

```
12270                      'reason': set([('list_head', 'prev'),
12271                                     ('mm_segment_t', 'seg'),
12272                                     ('task_struct', 'flags')])},
12273                    {'call': 'fchown',
12274                     'reason': set([('fd', 'file'),
12275                                    ('fd', 'flags'),
12276                                    ('list_head', 'prev')])},
12277                    {'call': 'ioperm',
12278                     'reason': set([('mm_segment_t', 'seg')])},
12279                    {'call': 'mq_timedreceive',
12280                     'reason': set([('fd', 'file'),
12281                                    ('fd', 'flags'),
12282                                    ('list_head', 'prev'),
12283                                    ('mm_segment_t', 'seg'),
12284                                    ('task_struct', 'flags')])},
12285                    {'call': 'capget',
12286                     'reason': set([('list_head', 'prev'),
12287                                    ('mm_segment_t', 'seg'),
12288                                    ('task_struct', 'flags')])},
12289                    {'call': 'utime',
12290                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12291                    {'call': 'sched_setaffinity',
12292                     'reason': set([('list_head', 'prev'),
12293                                    ('mm_segment_t', 'seg'),
12294                                    ('task_struct', 'flags')])},
12295                    {'call': 'ustat',
12296                     'reason': set([('list_head', 'prev')])},
12297                    {'call': 'fsync',
12298                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12299                    {'call': 'bpf',
12300                     'reason': set([('fd', 'file'),
12301                                    ('fd', 'flags'),
12302                                    ('list_head', 'prev')])},
12303                    {'call': 'unshare',
12304                     'reason': set([('list_head', 'prev')])},
12305                    {'call': 'signal',
12306                     'reason': set([('list_head', 'prev'),
12307                                    ('mm_segment_t', 'seg'),
12308                                    ('task_struct', 'flags')])},
12309                    {'call': 'setreuid',
12310                     'reason': set([('list_head', 'prev'),
12311                                    ('task_struct', 'flags')])},
12312                    {'call': 'semtimedop',
12313                     'reason': set([('list_head', 'prev'),
12314                                    ('mm_segment_t', 'seg'),
12315                                    ('task_struct', 'flags')])},
12316                    {'call': 'umount',
12317                     'reason': set([('list_head', 'prev'),
12318                                    ('mm_segment_t', 'seg'),
12319                                    ('task_struct', 'flags')])},
12320                    {'call': 'recvfrom',
12321                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12322                    {'call': 'fsetxattr',
12323                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12324                    {'call': 'timer_create',
12325                     'reason': set([('list_head', 'prev')])},
12326                    {'call': 'sendto',
12327                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12328                    {'call': 'mkdirat',
12329                     'reason': set([('list_head', 'prev')])},
12330                    {'call': 'sched_rr_get_interval',
12331                     'reason': set([('list_head', 'prev'),
12332                                    ('mm_segment_t', 'seg'),
12333                                    ('task_struct', 'flags')])},
12334                    {'call': 'epoll_create1',
12335                     'reason': set([('file', 'f_op'),
```



```
12336                                    ('list_head', 'prev')])},
12337                    {'call': 'timerfd_gettime',
12338                     'reason': set([('list_head', 'prev')])},
12339                    {'call': 'tee',
12340                     'reason': set([('fd', 'file'),
12341                                    ('fd', 'flags'),
12342                                    ('list_head', 'prev')])},
12343                    {'call': 'semctl',
12344                     'reason': set([('list_head', 'prev')])},
12345                    {'call': 'sync_file_range',
12346                     'reason': set([('fd', 'file'),
12347                                    ('fd', 'flags'),
12348                                    ('list_head', 'prev')])},
12349                    {'call': 'lseek',
12350                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12351                    {'call': 'connect',
12352                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12353                    {'call': 'getsockname',
12354                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12355                    {'call': 'epoll_ctl',
12356                     'reason': set([('fd', 'file'),
12357                                    ('fd', 'flags'),
12358                                    ('file', 'f_op'),
12359                                    ('list_head', 'prev')])},
12360                    {'call': 'flock',
12361                     'reason': set([('fd', 'file'),
12362                                    ('fd', 'flags'),
12363                                    ('file', 'f_op'),
12364                                    ('list_head', 'prev')])},
12365                    {'call': 'pwritev',
12366                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12367                    {'call': 'fchdir',
12368                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
12369                    {'call': 'openat',
12370                     'reason': set([('file', 'f_op'),
12371                                    ('list_head', 'prev')])},
12372                    {'call': 'lookup_dcookie',
12373                     'reason': set([('list_head', 'prev')])},
12374                    {'call': 'uselib',
12375                     'reason': set([('file', 'f_op'),
12376                                    ('list_head', 'prev')])},
12377                    {'call': 'renameat2',
12378                     'reason': set([('list_head', 'prev')])},
12379                    {'call': 'rt_sigprocmask',
12380                     'reason': set([('list_head', 'prev'),
12381                                    ('mm_segment_t', 'seg'),
12382                                    ('task_struct', 'flags')])},
12383                    {'call': 'accept4',
12384                     'reason': set([('fd', 'file'),
12385                                    ('fd', 'flags'),
12386                                    ('file', 'f_op'),
12387                                    ('list_head', 'prev')])},
12388                    {'call': 'msgctl',
12389                     'reason': set([('list_head', 'prev')])},
12390                    {'call': 'reboot',
12391                     'reason': set([('list_head', 'prev')])},
12392                    {'call': 'setsid',
12393                     'reason': set([('list_head', 'prev'),
12394                                    ('mm_segment_t', 'seg'),
12395                                    ('task_struct', 'flags')])},
12396                    {'call': 'set_trip_temp',
12397                     'reason': set([('list_head', 'prev')])},
12398                    {'call': 'sigaltstack',
12399                     'reason': set([('list_head', 'prev'),
12400                                    ('mm_segment_t', 'seg'),
12401                                    ('task_struct', 'flags')])},
```

```
12402                     {'call': 'sched_setattr',
12403                      'reason': set([('list_head', 'prev'),
12404                                     ('mm_segment_t', 'seg'),
12405                                     ('task_struct', 'flags')])},
12406                     {'call': 'old_readdir',
12407                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12408                     {'call': 'inotify_rm_watch',
12409                      'reason': set([('fd', 'file'),
12410                                     ('fd', 'flags'),
12411                                     ('list_head', 'prev')])},
12412                     {'call': 'socketpair',
12413                      'reason': set([('file', 'f_op'),
12414                                     ('list_head', 'prev')])},
12415                     {'call': 'utimensat',
12416                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12417                     {'call': 'migrate_pages',
12418                      'reason': set([('list_head', 'prev'),
12419                                     ('mm_segment_t', 'seg'),
12420                                     ('task_struct', 'flags')])},
12421                     {'call': 'getitimer',
12422                      'reason': set([('list_head', 'prev'),
12423                                     ('mm_segment_t', 'seg'),
12424                                     ('task_struct', 'flags')])},
12425                     {'call': 'fchmodat',
12426                      'reason': set([('list_head', 'prev')])},
12427                     {'call': 'setpgid',
12428                      'reason': set([('list_head', 'prev'),
12429                                     ('mm_segment_t', 'seg'),
12430                                     ('task_struct', 'flags')])},
12431                     {'call': 'init_module',
12432                      'reason': set([('list_head', 'prev')])},
12433                     {'call': 'setresgid',
12434                      'reason': set([('list_head', 'prev')])},
12435                     {'call': 'getcwd',
12436                      'reason': set([('list_head', 'prev')])},
12437                     {'call': 'inotify_add_watch',
12438                      'reason': set([('fd', 'file'),
12439                                     ('fd', 'flags'),
12440                                     ('list_head', 'prev')])},
12441                     {'call': 'get_trip_temp',
12442                      'reason': set([('list_head', 'prev')])},
12443                     {'call': 'preadv2',
12444                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12445                     {'call': 'timer_settime',
12446                      'reason': set([('list_head', 'prev')])},
12447                     {'call': 'setregid',
12448                      'reason': set([('list_head', 'prev')])},
12449                     {'call': 'splice',
12450                      'reason': set([('fd', 'file'),
12451                                     ('fd', 'flags'),
12452                                     ('list_head', 'prev')])},
12453                     {'call': 'ftruncate',
12454                      'reason': set([('fd', 'file'),
12455                                     ('fd', 'flags'),
12456                                     ('list_head', 'prev')])},
12457                     {'call': 'timer_gettime',
12458                      'reason': set([('list_head', 'prev')])},
12459                     {'call': 'preadv',
12460                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12461                     {'call': 'getpeername',
12462                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12463                     {'call': 'getsid',
12464                      'reason': set([('list_head', 'prev'),
12465                                     ('mm_segment_t', 'seg'),
12466                                     ('task_struct', 'flags')])},
12467                     {'call': 'shmat',
```



```
12468                      'reason': set([('file', 'f_op'),
12469                                     ('list_head', 'prev')])},
12470                     {'call': 'setsockopt',
12471                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12472                     {'call': 'mknodat',
12473                      'reason': set([('list_head', 'prev')])},
12474                     {'call': 'socket',
12475                      'reason': set([('file', 'f_op'),
12476                                     ('list_head', 'prev')])},
12477                     {'call': 'symlinkat',
12478                      'reason': set([('list_head', 'prev')])},
12479                     {'call': 'pipe2',
12480                      'reason': set([('file', 'f_op'),
12481                                     ('list_head', 'prev')])},
12482                     {'call': 'fcntl',
12483                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12484                     {'call': 'ioctl',
12485                      'reason': set([('fd', 'file'),
12486                                     ('fd', 'flags'),
12487                                     ('list_head', 'prev')])},
12488                     {'call': 'prlimit64',
12489                      'reason': set([('list_head', 'prev'),
12490                                     ('mm_segment_t', 'seg'),
12491                                     ('task_struct', 'flags')])},
12492                     {'call': 'pwrite64',
12493                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12494                     {'call': 'perf_event_open',
12495                      'reason': set([('fd', 'file'),
12496                                     ('fd', 'flags'),
12497                                     ('file', 'f_op'),
12498                                     ('hw_perf_event', 'sample_period'),
12499                                     ('list_head', 'prev'),
12500                                     ('mm_segment_t', 'seg'),
12501                                     ('perf_cpu_context', 'online'),
12502                                     ('perf_event', 'addr_filters_offs'),
12503                                     ('perf_event', 'attach_state'),
12504                                     ('perf_event', 'clock'),
12505                                     ('perf_event', 'cpu'),
12506                                     ('perf_event', 'ctx'),
12507                                     ('perf_event', 'destroy'),
12508                                     ('perf_event', 'event_caps'),
12509                                     ('perf_event', 'group_caps'),
12510                                     ('perf_event', 'group_leader'),
12511                                     ('perf_event', 'header_size'),
12512                                     ('perf_event', 'id_header_size'),
12513                                     ('perf_event', 'nr_siblings'),
12514                                     ('perf_event', 'ns'),
12515                                     ('perf_event', 'parent'),
12516                                     ('perf_event', 'pmu'),
12517                                     ('perf_event', 'prog'),
12518                                     ('perf_event', 'read_size'),
12519                                     ('perf_event_attr', '__reserved_1'),
12520                                     ('perf_event_attr', 'disabled'),
12521                                     ('perf_event_attr', 'exclude_hv'),
12522                                     ('perf_event_attr', 'exclude_kernel'),
12523                                     ('perf_event_attr', 'exclude_user'),
12524                                     ('perf_event_attr', 'exclusive'),
12525                                     ('perf_event_attr', 'freq'),
12526                                     ('perf_event_attr', 'inherit'),
12527                                     ('perf_event_attr', 'namespaces'),
12528                                     ('perf_event_attr', 'pinned'),
12529                                     ('perf_event_attr', 'read_format'),
12530                                     ('perf_event_attr', 'sample_freq'),
12531                                     ('perf_event_attr',
12532                                      'sample_max_stack'),
12533                                     ('perf_event_attr', 'sample_period'),
```

```
                                  ('perf_event_attr',
                                   'sample_regs_intr'),
                                  ('perf_event_attr',
                                   'sample_regs_user'),
                                  ('perf_event_attr',
                                   'sample_stack_user'),
                                  ('perf_event_attr', 'sample_type'),
                                  ('perf_event_attr', 'use_clockid'),
                                  ('perf_event_context', 'parent_ctx'),
                                  ('perf_event_context', 'task'),
                                  ('perf_event_context',
                                   'task_ctx_data'),
                                  ('pmu', 'capabilities'),
                                  ('pmu', 'module'),
                                  ('pmu', 'nr_addr_filters'),
                                  ('pmu', 'task_ctx_nr'),
                                  ('pmu', 'task_ctx_size'),
                                  ('task_struct', 'flags')])},
                 {'call': 'linkat',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'getgroups16',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'shmdt',
                  'reason': set([('file', 'f_op'),
                                 ('list_head', 'prev')])},
                 {'call': 'pwritev64v2',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'quotactl',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'rt_sigaction',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'futimesat',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'request_key',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'getpgid',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'brk', 'reason': set([('list_head', 'prev')])},
                 {'call': 'pwritev2',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'shutdown',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'acct',
                  'reason': set([('file', 'f_op'),
                                 ('list_head', 'prev')])},
                 {'call': 'open',
                  'reason': set([('file', 'f_op'),
                                 ('list_head', 'prev')])},
                 {'call': 'unlink',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'getsockopt',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'exit_group',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'getpriority',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'sigaction',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
```



```
                 {'call': 'mq_getsetattr',
                  'reason': set([('fd', 'file'),
                                 ('fd', 'flags'),
                                 ('list_head', 'prev')])},
                 {'call': 'faccessat',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'rmdir',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'dup',
                  'reason': set([('file', 'f_op'),
                                 ('list_head', 'prev')])},
                 {'call': 'fdatasync',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'setgroups16',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'setns',
                  'reason': set([('file', 'f_op'),
                                 ('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'getdents64',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'listen',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'fork',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'get_mempolicy',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'io_submit',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'get_robust_list',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'copy_file_range',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'mq_timedsend',
                  'reason': set([('fd', 'file'),
                                 ('fd', 'flags'),
                                 ('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'sched_yield',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'sched_getscheduler',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'fgetxattr',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'ptrace',
                  'reason': set([('list_head', 'prev'),
                                 ('mm_segment_t', 'seg'),
                                 ('task_struct', 'flags')])},
                 {'call': 'shmctl',
                  'reason': set([('file', 'f_op'),
                                 ('list_head', 'prev')])},
                 {'call': 'fcntl64',
                  'reason': set([('fd', 'file'), ('fd', 'flags')])},
                 {'call': 'munlockall',
                  'reason': set([('list_head', 'prev')])},
                 {'call': 'swapon',
                  'reason': set([('file', 'f_op'),
                                 ('list_head', 'prev')])},
```

```
12666                     {'call': 'fallocate',
12667                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12668                     {'call': 'pkey_mprotect',
12669                      'reason': set([('list_head', 'prev')])},
12670                     {'call': 'madvise',
12671                      'reason': set([('list_head', 'prev')])},
12672                     {'call': 'epoll_wait',
12673                      'reason': set([('fd', 'file'),
12674                                     ('fd', 'flags'),
12675                                     ('list_head', 'prev')])},
12676                     {'call': 'sched_getattr',
12677                      'reason': set([('list_head', 'prev'),
12678                                     ('mm_segment_t', 'seg'),
12679                                     ('task_struct', 'flags')])},
12680                     {'call': 'fchownat',
12681                      'reason': set([('list_head', 'prev')])},
12682                     {'call': 'getrusage',
12683                      'reason': set([('list_head', 'prev'),
12684                                     ('mm_segment_t', 'seg'),
12685                                     ('task_struct', 'flags')])},
12686                     {'call': 'timerfd_settime',
12687                      'reason': set([('list_head', 'prev')])},
12688                     {'call': 'sched_setscheduler',
12689                      'reason': set([('list_head', 'prev'),
12690                                     ('mm_segment_t', 'seg'),
12691                                     ('task_struct', 'flags')])},
12692                     {'call': 'setresuid',
12693                      'reason': set([('list_head', 'prev'),
12694                                     ('task_struct', 'flags')])},
12695                     {'call': 'setitimer',
12696                      'reason': set([('list_head', 'prev'),
12697                                     ('mm_segment_t', 'seg'),
12698                                     ('task_struct', 'flags')])},
12699                     {'call': 'ioprio_get',
12700                      'reason': set([('list_head', 'prev'),
12701                                     ('mm_segment_t', 'seg'),
12702                                     ('task_struct', 'flags')])},
12703                     {'call': 'vfork',
12704                      'reason': set([('list_head', 'prev'),
12705                                     ('mm_segment_t', 'seg'),
12706                                     ('task_struct', 'flags')])},
12707                     {'call': 'setuid',
12708                      'reason': set([('list_head', 'prev'),
12709                                     ('task_struct', 'flags')])},
12710                     {'call': 'llseek',
12711                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12712                     {'call': 'io_setup',
12713                      'reason': set([('list_head', 'prev')])},
12714                     {'call': 'mprotect',
12715                      'reason': set([('list_head', 'prev')])},
12716                     {'call': 'mmap_pgoff',
12717                      'reason': set([('file', 'f_op'),
12718                                     ('list_head', 'prev')])},
12719                     {'call': 'mremap',
12720                      'reason': set([('list_head', 'prev')])},
12721                     {'call': 'io_destroy',
12722                      'reason': set([('list_head', 'prev')])},
12723                     {'call': 'mbind',
12724                      'reason': set([('list_head', 'prev')])},
12725                     {'call': 'preadv64v2',
12726                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12727                     {'call': 'readv',
12728                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12729                     {'call': 'prctl',
12730                      'reason': set([('list_head', 'prev'),
12731                                     ('mm_segment_t', 'seg'),
```



```
12732                                     ('task_struct', 'flags')])},
12733                     {'call': 'move_pages',
12734                      'reason': set([('list_head', 'prev'),
12735                                     ('mm_segment_t', 'seg'),
12736                                     ('task_struct', 'flags')])},
12737                     {'call': 'timerfd_create',
12738                      'reason': set([('list_head', 'prev')])},
12739                     {'call': 'fstatfs',
12740                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12741                     {'call': 'modify_ldt',
12742                      'reason': set([('list_head', 'prev')])},
12743                     {'call': 'getgroups',
12744                      'reason': set([('list_head', 'prev')])},
12745                     {'call': 'fstatfs64',
12746                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12747                     {'call': 'dup2', 'reason': set([('list_head', 'prev')])},
12748                     {'call': 'get_curr_temp',
12749                      'reason': set([('list_head', 'prev')])},
12750                     {'call': 'msgsnd',
12751                      'reason': set([('list_head', 'prev')])},
12752                     {'call': 'write',
12753                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
12754                     {'call': 'munlock',
12755                      'reason': set([('list_head', 'prev')])},
12756                     {'call': 'setpriority',
12757                      'reason': set([('list_head', 'prev'),
12758                                     ('mm_segment_t', 'seg'),
12759                                     ('task_struct', 'flags')])},
12760                     {'call': 'inotify_init1',
12761                      'reason': set([('list_head', 'prev')])},
12762                     {'call': 'mincore',
12763                      'reason': set([('list_head', 'prev')])},
12764                     {'call': 'mq_notify',
12765                      'reason': set([('fd', 'file'),
12766                                     ('fd', 'flags'),
12767                                     ('list_head', 'prev')])},
12768                     {'call': 'sendfile',
12769                      'reason': set([('fd', 'file'),
12770                                     ('fd', 'flags'),
12771                                     ('list_head', 'prev')])},
12772                     {'call': 'timer_getoverrun',
12773                      'reason': set([('list_head', 'prev')])},
12774                     {'call': 'kexec_load',
12775                      'reason': set([('list_head', 'prev')])},
12776                     {'call': 'clone',
12777                      'reason': set([('list_head', 'prev'),
12778                                     ('mm_segment_t', 'seg'),
12779                                     ('task_struct', 'flags')])},
12780                     {'call': 'mq_open',
12781                      'reason': set([('file', 'f_op'),
12782                                     ('list_head', 'prev')])},
12783                     {'call': 'setgroups',
12784                      'reason': set([('list_head', 'prev')])},
12785                     {'call': 'unlinkat',
12786                      'reason': set([('list_head', 'prev')])},
12787                     {'call': 'sched_getparam',
12788                      'reason': set([('list_head', 'prev'),
12789                                     ('mm_segment_t', 'seg'),
12790                                     ('task_struct', 'flags')])},
12791                     {'call': 'io_cancel',
12792                      'reason': set([('list_head', 'prev')])},
12793                     {'call': 'open_by_handle_at',
12794                      'reason': set([('file', 'f_op'),
12795                                     ('list_head', 'prev')])},
12796                     {'call': 'bind',
12797                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
                    {'call': 'flistxattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'finit_module',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sendfile64',
                     'reason': set([('fd', 'file'),
                                    ('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'mlockall',
                     'reason': set([('list_head', 'prev')])}],
 'pivot_root': [{'call': 'eventfd2',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mq_unlink',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'swapoff',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'pivot_root',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent'),
                                ('mount', 'mnt_ns'),
                                ('mount', 'mnt_parent'),
                                ('path', 'dentry'),
                                ('path', 'mnt'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'memfd_create',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'remap_file_pages',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'dup3',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'unshare',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'umount',
                 'reason': set([('mount', 'mnt_ns'),
                                ('mount', 'mnt_parent'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'mkdirat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'epoll_create1',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'epoll_ctl',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'flock',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'openat',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'lookup_dcookie',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'uselib',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'renameat2',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'accept4',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'socketpair',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'getcwd',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent'),
```



```
                                ('mount', 'mnt_ns'),
                                ('mount', 'mnt_parent'),
                                ('path', 'dentry'),
                                ('path', 'mnt'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'ftruncate',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'shmat',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mknodat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'socket',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'symlinkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'pipe2',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'perf_event_open',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'linkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'shmdt',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'quotactl',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'acct',
                 'reason': set([('path', 'dentry'),
                                ('path', 'mnt'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'open',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'unlink',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'rmdir',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'dup',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'setns',
                 'reason': set([('nsproxy', 'mnt_ns'),
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'shmctl',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'swapon',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mmap_pgoff',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mq_open',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent'),
                                ('path', 'dentry'),
                                ('path', 'mnt'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'unlinkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'open_by_handle_at',
```

```
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'pkey_alloc': [{'call': 'swapoff',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'pkey_alloc',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'remap_file_pages',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_getevents',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'pkey_free',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'migrate_pages',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'shmdt',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'brk',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'get_mempolicy',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'getrusage',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_setup',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'mremap',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_destroy',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'mbind',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'prctl',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'move_pages',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'modify_ldt',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'mincore',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_cancel',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])}],
 'pkey_mprotect': [{'call': 'keyctl',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'msgrcv',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'kill',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sched_setparam',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'ioprio_set',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'personality',
                    'reason': set([('task_struct', 'personality')])},
                   {'call': 'remap_file_pages',
                    'reason': set([('vm_area_struct', 'vm_end'),
                                   ('vm_area_struct', 'vm_flags'),
                                   ('vm_area_struct', 'vm_start')])},
```



```
                   {'call': 'getppid',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'capget',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sched_setaffinity',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'signal',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'semtimedop',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'umount',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sched_rr_get_interval',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'rt_sigprocmask',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'setsid',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sigaltstack',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sched_setattr',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'migrate_pages',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'getitimer',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'setpgid',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'getsid',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'prlimit64',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'perf_event_open',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'shmdt',
                    'reason': set([('vm_area_struct', 'vm_end'),
                                   ('vm_area_struct', 'vm_flags'),
                                   ('vm_area_struct', 'vm_start')])},
                   {'call': 'rt_sigaction',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'getpgid',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'brk',
                    'reason': set([('vm_area_struct', 'vm_end'),
```

```
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'getpriority',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'sigaction',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'setns',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'fork',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'get_mempolicy',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'get_robust_list',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'ptrace',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'munlockall',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'pkey_mprotect',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'madvise',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'sched_getattr',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'getrusage',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'setitimer',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'ioprio_get',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'vfork',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'mprotect',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'mremap',
                     'reason': set([('vm_area_struct', 'vm_end'),
```



```
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'prctl',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality'),
                                    ('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'move_pages',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'munlock',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'setpriority',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'mincore',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'clone',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'sched_getparam',
                     'reason': set([('task_struct', 'mm'),
                                    ('task_struct', 'personality')])},
                    {'call': 'mlockall',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])}],
 'poll': [{'call': 'poll', 'reason': set([('poll_list', 'len')])},
          {'call': 'ppoll', 'reason': set([('poll_list', 'len')])}],
 'ppoll': [{'call': 'keyctl',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('task_struct', 'personality'),
                           ('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'msgrcv',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'mq_unlink',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
           {'call': 'swapoff',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'sched_getaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setparam',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'fchmod',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'memfd_create',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'ioprio_set',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'personality',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'readlinkat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'io_getevents',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'getppid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'fchown',
```

```
13194             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13195            {'call': 'mq_timedreceive',
13196             'reason': set([('task_struct', 'personality'),
13197                            ('timespec', 'tv_nsec'),
13198                            ('timespec', 'tv_sec')])},
13199            {'call': 'capget',
13200             'reason': set([('task_struct', 'personality')])},
13201            {'call': 'utime',
13202             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13203            {'call': 'sched_setaffinity',
13204             'reason': set([('task_struct', 'personality')])},
13205            {'call': 'signal',
13206             'reason': set([('task_struct', 'personality')])},
13207            {'call': 'semtimedop',
13208             'reason': set([('task_struct', 'personality'),
13209                            ('timespec', 'tv_nsec'),
13210                            ('timespec', 'tv_sec')])},
13211            {'call': 'umount',
13212             'reason': set([('task_struct', 'personality')])},
13213            {'call': 'settimeofday',
13214             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13215            {'call': 'sched_rr_get_interval',
13216             'reason': set([('task_struct', 'personality'),
13217                            ('timespec', 'tv_nsec'),
13218                            ('timespec', 'tv_sec')])},
13219            {'call': 'timerfd_gettime',
13220             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13221            {'call': 'pselect6',
13222             'reason': set([('compat_timespec', 'tv_nsec'),
13223                            ('compat_timespec', 'tv_sec'),
13224                            ('timespec', 'tv_nsec'),
13225                            ('timespec', 'tv_sec')])},
13226            {'call': 'uselib',
13227             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13228            {'call': 'rt_sigprocmask',
13229             'reason': set([('task_struct', 'personality')])},
13230            {'call': 'setsid',
13231             'reason': set([('task_struct', 'personality')])},
13232            {'call': 'sigaltstack',
13233             'reason': set([('task_struct', 'personality')])},
13234            {'call': 'sched_setattr',
13235             'reason': set([('task_struct', 'personality')])},
13236            {'call': 'migrate_pages',
13237             'reason': set([('task_struct', 'personality')])},
13238            {'call': 'getitimer',
13239             'reason': set([('task_struct', 'personality')])},
13240            {'call': 'fchmodat',
13241             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13242            {'call': 'setpgid',
13243             'reason': set([('task_struct', 'personality')])},
13244            {'call': 'inotify_add_watch',
13245             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13246            {'call': 'timer_settime',
13247             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13248            {'call': 'ftruncate',
13249             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13250            {'call': 'timer_gettime',
13251             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13252            {'call': 'getsid',
13253             'reason': set([('task_struct', 'personality')])},
13254            {'call': 'ioctl',
13255             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13256            {'call': 'prlimit64',
13257             'reason': set([('task_struct', 'personality')])},
13258            {'call': 'perf_event_open',
13259             'reason': set([('task_struct', 'personality')])},
```



```
13260            {'call': 'linkat',
13261             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13262            {'call': 'stime',
13263             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13264            {'call': 'rt_sigaction',
13265             'reason': set([('task_struct', 'personality')])},
13266            {'call': 'futimesat',
13267             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13268            {'call': 'getpgid',
13269             'reason': set([('task_struct', 'personality')])},
13270            {'call': 'poll',
13271             'reason': set([('poll_list', 'len'),
13272                            ('timespec', 'tv_nsec'),
13273                            ('timespec', 'tv_sec')])},
13274            {'call': 'select',
13275             'reason': set([('compat_timespec', 'tv_nsec'),
13276                            ('compat_timespec', 'tv_sec'),
13277                            ('timespec', 'tv_nsec'),
13278                            ('timespec', 'tv_sec')])},
13279            {'call': 'unlink',
13280             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13281            {'call': 'getpriority',
13282             'reason': set([('task_struct', 'personality')])},
13283            {'call': 'sigaction',
13284             'reason': set([('task_struct', 'personality')])},
13285            {'call': 'nanosleep',
13286             'reason': set([('compat_timespec', 'tv_nsec'),
13287                            ('compat_timespec', 'tv_sec'),
13288                            ('timespec', 'tv_nsec'),
13289                            ('timespec', 'tv_sec')])},
13290            {'call': 'mq_getsetattr',
13291             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13292            {'call': 'faccessat',
13293             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13294            {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
13295            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
13296            {'call': 'get_robust_list',
13297             'reason': set([('task_struct', 'personality')])},
13298            {'call': 'mq_timedsend',
13299             'reason': set([('task_struct', 'personality'),
13300                            ('timespec', 'tv_nsec'),
13301                            ('timespec', 'tv_sec')])},
13302            {'call': 'sched_getscheduler',
13303             'reason': set([('task_struct', 'personality')])},
13304            {'call': 'ptrace',
13305             'reason': set([('task_struct', 'personality')])},
13306            {'call': 'swapon',
13307             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13308            {'call': 'epoll_wait',
13309             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13310            {'call': 'sched_getattr',
13311             'reason': set([('task_struct', 'personality')])},
13312            {'call': 'fchownat',
13313             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13314            {'call': 'getrusage',
13315             'reason': set([('task_struct', 'personality')])},
13316            {'call': 'fstat',
13317             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13318            {'call': 'timerfd_settime',
13319             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13320            {'call': 'sched_setscheduler',
13321             'reason': set([('task_struct', 'personality')])},
13322            {'call': 'setitimer',
13323             'reason': set([('task_struct', 'personality')])},
13324            {'call': 'ioprio_get',
13325             'reason': set([('task_struct', 'personality')])},
```

```
13326           {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
13327           {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
13328           {'call': 'move_pages',
13329            'reason': set([('task_struct', 'personality')])},
13330           {'call': 'setpriority',
13331            'reason': set([('task_struct', 'personality')])},
13332           {'call': 'mq_notify',
13333            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13334           {'call': 'sendfile',
13335            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13336           {'call': 'newfstat',
13337            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13338           {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
13339           {'call': 'clock_nanosleep',
13340            'reason': set([('compat_timespec', 'tv_nsec'),
13341                           ('compat_timespec', 'tv_sec'),
13342                           ('timespec', 'tv_nsec'),
13343                           ('timespec', 'tv_sec')])},
13344           {'call': 'unlinkat',
13345            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13346           {'call': 'sched_getparam',
13347            'reason': set([('task_struct', 'personality')])},
13348           {'call': 'futex',
13349            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13350           {'call': 'recvmmsg',
13351            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13352           {'call': 'sendfile64',
13353            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
13354           {'call': 'ppoll',
13355            'reason': set([('compat_timespec', 'tv_nsec'),
13356                           ('compat_timespec', 'tv_sec'),
13357                           ('poll_list', 'len'),
13358                           ('timespec', 'tv_nsec'),
13359                           ('timespec', 'tv_sec')])}],
13360  'prctl': [{'call': 'keyctl',
13361             'reason': set([('task_struct', 'flags'),
13362                            ('task_struct', 'personality'),
13363                            ('task_struct', 'timer_slack_ns')])},
13364           {'call': 'rt_sigtimedwait',
13365            'reason': set([('task_struct', 'flags'),
13366                           ('task_struct', 'personality'),
13367                           ('task_struct', 'timer_slack_ns')])},
13368           {'call': 'msgrcv',
13369            'reason': set([('task_struct', 'flags'),
13370                           ('task_struct', 'personality'),
13371                           ('task_struct', 'timer_slack_ns')])},
13372           {'call': 'kill',
13373            'reason': set([('task_struct', 'flags'),
13374                           ('task_struct', 'personality'),
13375                           ('task_struct', 'timer_slack_ns')])},
13376           {'call': 'swapoff', 'reason': set([('mm_struct', 'flags')])},
13377           {'call': 'sched_getaffinity',
13378            'reason': set([('task_struct', 'flags'),
13379                           ('task_struct', 'personality'),
13380                           ('task_struct', 'timer_slack_ns')])},
13381           {'call': 'sched_setparam',
13382            'reason': set([('task_struct', 'flags'),
13383                           ('task_struct', 'personality'),
13384                           ('task_struct', 'timer_slack_ns')])},
13385           {'call': 'ioprio_set',
13386            'reason': set([('task_struct', 'flags'),
13387                           ('task_struct', 'personality'),
13388                           ('task_struct', 'timer_slack_ns')])},
13389           {'call': 'personality',
13390            'reason': set([('task_struct', 'personality')])},
13391           {'call': 'remap_file_pages',
```



```
13392            'reason': set([('mm_struct', 'flags')])},
13393           {'call': 'io_getevents', 'reason': set([('mm_struct', 'flags')])},
13394           {'call': 'getppid',
13395            'reason': set([('task_struct', 'flags'),
13396                           ('task_struct', 'personality'),
13397                           ('task_struct', 'timer_slack_ns')])},
13398           {'call': 'mq_timedreceive',
13399            'reason': set([('task_struct', 'flags'),
13400                           ('task_struct', 'personality'),
13401                           ('task_struct', 'timer_slack_ns')])},
13402           {'call': 'capget',
13403            'reason': set([('task_struct', 'flags'),
13404                           ('task_struct', 'personality'),
13405                           ('task_struct', 'timer_slack_ns')])},
13406           {'call': 'sched_setaffinity',
13407            'reason': set([('task_struct', 'flags'),
13408                           ('task_struct', 'personality'),
13409                           ('task_struct', 'timer_slack_ns')])},
13410           {'call': 'signal',
13411            'reason': set([('task_struct', 'flags'),
13412                           ('task_struct', 'personality'),
13413                           ('task_struct', 'timer_slack_ns')])},
13414           {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
13415           {'call': 'semtimedop',
13416            'reason': set([('task_struct', 'flags'),
13417                           ('task_struct', 'personality'),
13418                           ('task_struct', 'timer_slack_ns')])},
13419           {'call': 'umount',
13420            'reason': set([('task_struct', 'flags'),
13421                           ('task_struct', 'personality'),
13422                           ('task_struct', 'timer_slack_ns')])},
13423           {'call': 'sched_rr_get_interval',
13424            'reason': set([('task_struct', 'flags'),
13425                           ('task_struct', 'personality'),
13426                           ('task_struct', 'timer_slack_ns')])},
13427           {'call': 'rt_sigprocmask',
13428            'reason': set([('task_struct', 'flags'),
13429                           ('task_struct', 'personality'),
13430                           ('task_struct', 'timer_slack_ns')])},
13431           {'call': 'setsid',
13432            'reason': set([('task_struct', 'flags'),
13433                           ('task_struct', 'personality'),
13434                           ('task_struct', 'timer_slack_ns')])},
13435           {'call': 'sigaltstack',
13436            'reason': set([('task_struct', 'flags'),
13437                           ('task_struct', 'personality'),
13438                           ('task_struct', 'timer_slack_ns')])},
13439           {'call': 'sched_setattr',
13440            'reason': set([('task_struct', 'flags'),
13441                           ('task_struct', 'personality'),
13442                           ('task_struct', 'timer_slack_ns')])},
13443           {'call': 'migrate_pages',
13444            'reason': set([('mm_struct', 'flags'),
13445                           ('task_struct', 'flags'),
13446                           ('task_struct', 'personality'),
13447                           ('task_struct', 'timer_slack_ns')])},
13448           {'call': 'getitimer',
13449            'reason': set([('task_struct', 'flags'),
13450                           ('task_struct', 'personality'),
13451                           ('task_struct', 'timer_slack_ns')])},
13452           {'call': 'setpgid',
13453            'reason': set([('task_struct', 'flags'),
13454                           ('task_struct', 'personality'),
13455                           ('task_struct', 'timer_slack_ns')])},
13456           {'call': 'getsid',
13457            'reason': set([('task_struct', 'flags'),
```

```
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'prlimit64',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'perf_event_open',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'shmdt', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'rt_sigaction',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'getpgid',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'brk', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'getpriority',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'sigaction',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'setns',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'fork',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'get_mempolicy', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'get_robust_list',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'mq_timedsend',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_getscheduler',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'ptrace',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_getattr',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'getrusage',
           'reason': set([('mm_struct', 'flags'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_setscheduler',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
```



```
          {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
          {'call': 'setitimer',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'ioprio_get',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'vfork',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
          {'call': 'io_setup', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'mremap', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'io_destroy', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'mbind', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'prctl',
           'reason': set([('mm_struct', 'flags'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'move_pages',
           'reason': set([('mm_struct', 'flags'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'modify_ldt', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'setpriority',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'mincore', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'clone',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_getparam',
           'reason': set([('task_struct', 'flags'),
                            ('task_struct', 'personality'),
                            ('task_struct', 'timer_slack_ns')])},
          {'call': 'io_cancel', 'reason': set([('mm_struct', 'flags')])}],
 'pread64': [{'call': 'syncfs',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'vmsplice',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
             {'call': 'pwritev64',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
             {'call': 'fremovexattr',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'readahead',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'getdents',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'writev',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'preadv64',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'fchmod',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'pread64',
              'reason': set([('fd', 'file'), ('fd', 'flags')])},
             {'call': 'signalfd4',
```

```
13590                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13591               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
13592               {'call': 'remap_file_pages',
13593                'reason': set([('file', 'f_mode')])},
13594               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
13595               {'call': 'read',
13596                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13597               {'call': 'fchown',
13598                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13599               {'call': 'mq_timedreceive',
13600                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13601               {'call': 'utime',
13602                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13603               {'call': 'fsync',
13604                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13605               {'call': 'bpf',
13606                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13607               {'call': 'recvfrom',
13608                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13609               {'call': 'fsetxattr',
13610                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13611               {'call': 'sendto',
13612                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13613               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
13614               {'call': 'tee',
13615                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13616               {'call': 'sync_file_range',
13617                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13618               {'call': 'lseek',
13619                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13620               {'call': 'connect',
13621                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13622               {'call': 'getsockname',
13623                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13624               {'call': 'epoll_ctl',
13625                'reason': set([('fd', 'file'),
13626                               ('fd', 'flags'),
13627                               ('file', 'f_mode')])},
13628               {'call': 'flock',
13629                'reason': set([('fd', 'file'),
13630                               ('fd', 'flags'),
13631                               ('file', 'f_mode')])},
13632               {'call': 'pwritev',
13633                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13634               {'call': 'fchdir',
13635                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13636               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
13637               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
13638               {'call': 'accept4',
13639                'reason': set([('fd', 'file'),
13640                               ('fd', 'flags'),
13641                               ('file', 'f_mode')])},
13642               {'call': 'old_readdir',
13643                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13644               {'call': 'inotify_rm_watch',
13645                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13646               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
13647               {'call': 'utimensat',
13648                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13649               {'call': 'inotify_add_watch',
13650                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13651               {'call': 'preadv2',
13652                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13653               {'call': 'splice',
13654                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13655               {'call': 'ftruncate',
```



```
13656                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13657               {'call': 'preadv',
13658                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13659               {'call': 'getpeername',
13660                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13661               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
13662               {'call': 'setsockopt',
13663                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13664               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
13665               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
13666               {'call': 'fcntl',
13667                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13668               {'call': 'ioctl',
13669                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13670               {'call': 'pwrite64',
13671                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13672               {'call': 'perf_event_open',
13673                'reason': set([('fd', 'file'),
13674                               ('fd', 'flags'),
13675                               ('file', 'f_mode')])},
13676               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
13677               {'call': 'pwritev64v2',
13678                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13679               {'call': 'futimesat',
13680                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13681               {'call': 'pwritev2',
13682                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13683               {'call': 'shutdown',
13684                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13685               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
13686               {'call': 'open', 'reason': set([('file', 'f_mode')])},
13687               {'call': 'getsockopt',
13688                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13689               {'call': 'mq_getsetattr',
13690                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13691               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
13692               {'call': 'fdatasync',
13693                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13694               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
13695               {'call': 'getdents64',
13696                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13697               {'call': 'listen',
13698                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13699               {'call': 'copy_file_range',
13700                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13701               {'call': 'mq_timedsend',
13702                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13703               {'call': 'fgetxattr',
13704                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13705               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
13706               {'call': 'fcntl64',
13707                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13708               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
13709               {'call': 'fallocate',
13710                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13711               {'call': 'epoll_wait',
13712                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13713               {'call': 'llseek',
13714                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13715               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
13716               {'call': 'preadv64v2',
13717                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13718               {'call': 'readv',
13719                'reason': set([('fd', 'file'), ('fd', 'flags')])},
13720               {'call': 'fstatfs',
13721                'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
13722             {'call': 'fstatfs64',
13723              'reason': set([('fd', 'file'), ('fd', 'flags')])},
13724             {'call': 'write',
13725              'reason': set([('fd', 'file'), ('fd', 'flags')])},
13726             {'call': 'mq_notify',
13727              'reason': set([('fd', 'file'), ('fd', 'flags')])},
13728             {'call': 'sendfile',
13729              'reason': set([('fd', 'file'), ('fd', 'flags')])},
13730             {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
13731             {'call': 'open_by_handle_at',
13732              'reason': set([('file', 'f_mode')])},
13733             {'call': 'bind',
13734              'reason': set([('fd', 'file'), ('fd', 'flags')])},
13735             {'call': 'flistxattr',
13736              'reason': set([('fd', 'file'), ('fd', 'flags')])},
13737             {'call': 'sendfile64',
13738              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
13739  'preadv': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
13740             {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
13741             {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
13742             {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
13743             {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
13744             {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
13745             {'call': 'readahead', 'reason': set([('fd', 'file')])},
13746             {'call': 'getdents', 'reason': set([('fd', 'file')])},
13747             {'call': 'writev', 'reason': set([('fd', 'file')])},
13748             {'call': 'preadv64', 'reason': set([('fd', 'file')])},
13749             {'call': 'fchmod', 'reason': set([('fd', 'file')])},
13750             {'call': 'pread64', 'reason': set([('fd', 'file')])},
13751             {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
13752             {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
13753             {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
13754             {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
13755             {'call': 'read', 'reason': set([('fd', 'file')])},
13756             {'call': 'fchown', 'reason': set([('fd', 'file')])},
13757             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
13758             {'call': 'utime', 'reason': set([('fd', 'file')])},
13759             {'call': 'fsync', 'reason': set([('fd', 'file')])},
13760             {'call': 'bpf', 'reason': set([('fd', 'file')])},
13761             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
13762             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
13763             {'call': 'sendto', 'reason': set([('fd', 'file')])},
13764             {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
13765             {'call': 'tee', 'reason': set([('fd', 'file')])},
13766             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
13767             {'call': 'lseek', 'reason': set([('fd', 'file')])},
13768             {'call': 'connect', 'reason': set([('fd', 'file')])},
13769             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
13770             {'call': 'epoll_ctl',
13771              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13772             {'call': 'flock',
13773              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13774             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
13775             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
13776             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
13777             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
13778             {'call': 'accept4',
13779              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13780             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
13781             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
13782             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
13783             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
13784             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
13785             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
13786             {'call': 'splice', 'reason': set([('fd', 'file')])},
13787             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
```



```
13788             {'call': 'preadv', 'reason': set([('fd', 'file')])},
13789             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
13790             {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
13791             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
13792             {'call': 'socket', 'reason': set([('file', 'f_mode')])},
13793             {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
13794             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
13795             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
13796             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
13797             {'call': 'perf_event_open',
13798              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13799             {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
13800             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
13801             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
13802             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
13803             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
13804             {'call': 'acct', 'reason': set([('file', 'f_mode')])},
13805             {'call': 'open', 'reason': set([('file', 'f_mode')])},
13806             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
13807             {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
13808             {'call': 'dup', 'reason': set([('file', 'f_mode')])},
13809             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
13810             {'call': 'setns', 'reason': set([('file', 'f_mode')])},
13811             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
13812             {'call': 'listen', 'reason': set([('fd', 'file')])},
13813             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
13814             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
13815             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
13816             {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
13817             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
13818             {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
13819             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
13820             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
13821             {'call': 'llseek', 'reason': set([('fd', 'file')])},
13822             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
13823             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
13824             {'call': 'readv', 'reason': set([('fd', 'file')])},
13825             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
13826             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
13827             {'call': 'write', 'reason': set([('fd', 'file')])},
13828             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
13829             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
13830             {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
13831             {'call': 'open_by_handle_at',
13832              'reason': set([('file', 'f_mode')])},
13833             {'call': 'bind', 'reason': set([('fd', 'file')])},
13834             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
13835             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
13836  'preadv2': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
13837              {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
13838              {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
13839              {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
13840              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
13841              {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
13842              {'call': 'readahead', 'reason': set([('fd', 'file')])},
13843              {'call': 'getdents', 'reason': set([('fd', 'file')])},
13844              {'call': 'writev', 'reason': set([('fd', 'file')])},
13845              {'call': 'preadv64', 'reason': set([('fd', 'file')])},
13846              {'call': 'fchmod', 'reason': set([('fd', 'file')])},
13847              {'call': 'pread64', 'reason': set([('fd', 'file')])},
13848              {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
13849              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
13850              {'call': 'remap_file_pages',
13851               'reason': set([('file', 'f_mode')])},
13852              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
13853              {'call': 'read', 'reason': set([('fd', 'file')])},
```

```
13854             {'call': 'fchown', 'reason': set([('fd', 'file')])},
13855             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
13856             {'call': 'utime', 'reason': set([('fd', 'file')])},
13857             {'call': 'fsync', 'reason': set([('fd', 'file')])},
13858             {'call': 'bpf', 'reason': set([('fd', 'file')])},
13859             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
13860             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
13861             {'call': 'sendto', 'reason': set([('fd', 'file')])},
13862             {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
13863             {'call': 'tee', 'reason': set([('fd', 'file')])},
13864             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
13865             {'call': 'lseek', 'reason': set([('fd', 'file')])},
13866             {'call': 'connect', 'reason': set([('fd', 'file')])},
13867             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
13868             {'call': 'epoll_ctl',
13869              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13870             {'call': 'flock',
13871              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13872             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
13873             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
13874             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
13875             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
13876             {'call': 'accept4',
13877              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13878             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
13879             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
13880             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
13881             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
13882             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
13883             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
13884             {'call': 'splice', 'reason': set([('fd', 'file')])},
13885             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
13886             {'call': 'preadv', 'reason': set([('fd', 'file')])},
13887             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
13888             {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
13889             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
13890             {'call': 'socket', 'reason': set([('file', 'f_mode')])},
13891             {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
13892             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
13893             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
13894             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
13895             {'call': 'perf_event_open',
13896              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13897             {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
13898             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
13899             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
13900             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
13901             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
13902             {'call': 'acct', 'reason': set([('file', 'f_mode')])},
13903             {'call': 'open', 'reason': set([('file', 'f_mode')])},
13904             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
13905             {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
13906             {'call': 'dup', 'reason': set([('file', 'f_mode')])},
13907             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
13908             {'call': 'setns', 'reason': set([('file', 'f_mode')])},
13909             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
13910             {'call': 'listen', 'reason': set([('fd', 'file')])},
13911             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
13912             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
13913             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
13914             {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
13915             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
13916             {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
13917             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
13918             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
13919             {'call': 'llseek', 'reason': set([('fd', 'file')])},
```



```
13920             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
13921             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
13922             {'call': 'readv', 'reason': set([('fd', 'file')])},
13923             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
13924             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
13925             {'call': 'write', 'reason': set([('fd', 'file')])},
13926             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
13927             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
13928             {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
13929             {'call': 'open_by_handle_at',
13930              'reason': set([('file', 'f_mode')])},
13931             {'call': 'bind', 'reason': set([('fd', 'file')])},
13932             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
13933             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
13934  'preadv64': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
13935               {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
13936               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
13937               {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
13938               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
13939               {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
13940               {'call': 'readahead', 'reason': set([('fd', 'file')])},
13941               {'call': 'getdents', 'reason': set([('fd', 'file')])},
13942               {'call': 'writev', 'reason': set([('fd', 'file')])},
13943               {'call': 'preadv64', 'reason': set([('fd', 'file')])},
13944               {'call': 'fchmod', 'reason': set([('fd', 'file')])},
13945               {'call': 'pread64', 'reason': set([('fd', 'file')])},
13946               {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
13947               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
13948               {'call': 'remap_file_pages',
13949                'reason': set([('file', 'f_mode')])},
13950               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
13951               {'call': 'read', 'reason': set([('fd', 'file')])},
13952               {'call': 'fchown', 'reason': set([('fd', 'file')])},
13953               {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
13954               {'call': 'utime', 'reason': set([('fd', 'file')])},
13955               {'call': 'fsync', 'reason': set([('fd', 'file')])},
13956               {'call': 'bpf', 'reason': set([('fd', 'file')])},
13957               {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
13958               {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
13959               {'call': 'sendto', 'reason': set([('fd', 'file')])},
13960               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
13961               {'call': 'tee', 'reason': set([('fd', 'file')])},
13962               {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
13963               {'call': 'lseek', 'reason': set([('fd', 'file')])},
13964               {'call': 'connect', 'reason': set([('fd', 'file')])},
13965               {'call': 'getsockname', 'reason': set([('fd', 'file')])},
13966               {'call': 'epoll_ctl',
13967                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13968               {'call': 'flock',
13969                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13970               {'call': 'pwritev', 'reason': set([('fd', 'file')])},
13971               {'call': 'fchdir', 'reason': set([('fd', 'file')])},
13972               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
13973               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
13974               {'call': 'accept4',
13975                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13976               {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
13977               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
13978               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
13979               {'call': 'utimensat', 'reason': set([('fd', 'file')])},
13980               {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
13981               {'call': 'preadv2', 'reason': set([('fd', 'file')])},
13982               {'call': 'splice', 'reason': set([('fd', 'file')])},
13983               {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
13984               {'call': 'preadv', 'reason': set([('fd', 'file')])},
13985               {'call': 'getpeername', 'reason': set([('fd', 'file')])},
```

```
13986               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
13987               {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
13988               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
13989               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
13990               {'call': 'fcntl', 'reason': set([('fd', 'file')])},
13991               {'call': 'ioctl', 'reason': set([('fd', 'file')])},
13992               {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
13993               {'call': 'perf_event_open',
13994                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
13995               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
13996               {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
13997               {'call': 'futimesat', 'reason': set([('fd', 'file')])},
13998               {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
13999               {'call': 'shutdown', 'reason': set([('fd', 'file')])},
14000               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
14001               {'call': 'open', 'reason': set([('file', 'f_mode')])},
14002               {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
14003               {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
14004               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
14005               {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
14006               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
14007               {'call': 'getdents64', 'reason': set([('fd', 'file')])},
14008               {'call': 'listen', 'reason': set([('fd', 'file')])},
14009               {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
14010               {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
14011               {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
14012               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
14013               {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
14014               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
14015               {'call': 'fallocate', 'reason': set([('fd', 'file')])},
14016               {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
14017               {'call': 'llseek', 'reason': set([('fd', 'file')])},
14018               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
14019               {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
14020               {'call': 'readv', 'reason': set([('fd', 'file')])},
14021               {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
14022               {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
14023               {'call': 'write', 'reason': set([('fd', 'file')])},
14024               {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
14025               {'call': 'sendfile', 'reason': set([('fd', 'file')])},
14026               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
14027               {'call': 'open_by_handle_at',
14028                'reason': set([('file', 'f_mode')])},
14029               {'call': 'bind', 'reason': set([('fd', 'file')])},
14030               {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
14031               {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
14032  'preadv64v2': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
14033                 {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
14034                 {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
14035                 {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
14036                 {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
14037                 {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
14038                 {'call': 'readahead', 'reason': set([('fd', 'file')])},
14039                 {'call': 'getdents', 'reason': set([('fd', 'file')])},
14040                 {'call': 'writev', 'reason': set([('fd', 'file')])},
14041                 {'call': 'preadv64', 'reason': set([('fd', 'file')])},
14042                 {'call': 'fchmod', 'reason': set([('fd', 'file')])},
14043                 {'call': 'pread64', 'reason': set([('fd', 'file')])},
14044                 {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
14045                 {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
14046                 {'call': 'remap_file_pages',
14047                  'reason': set([('file', 'f_mode')])},
14048                 {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
14049                 {'call': 'read', 'reason': set([('fd', 'file')])},
14050                 {'call': 'fchown', 'reason': set([('fd', 'file')])},
14051                 {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
```



```
14052               {'call': 'utime', 'reason': set([('fd', 'file')])},
14053               {'call': 'fsync', 'reason': set([('fd', 'file')])},
14054               {'call': 'bpf', 'reason': set([('fd', 'file')])},
14055               {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
14056               {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
14057               {'call': 'sendto', 'reason': set([('fd', 'file')])},
14058               {'call': 'epoll_create1',
14059                'reason': set([('file', 'f_mode')])},
14060               {'call': 'tee', 'reason': set([('fd', 'file')])},
14061               {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
14062               {'call': 'lseek', 'reason': set([('fd', 'file')])},
14063               {'call': 'connect', 'reason': set([('fd', 'file')])},
14064               {'call': 'getsockname', 'reason': set([('fd', 'file')])},
14065               {'call': 'epoll_ctl',
14066                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
14067               {'call': 'flock',
14068                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
14069               {'call': 'pwritev', 'reason': set([('fd', 'file')])},
14070               {'call': 'fchdir', 'reason': set([('fd', 'file')])},
14071               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
14072               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
14073               {'call': 'accept4',
14074                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
14075               {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
14076               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
14077               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
14078               {'call': 'utimensat', 'reason': set([('fd', 'file')])},
14079               {'call': 'inotify_add_watch',
14080                'reason': set([('fd', 'file')])},
14081               {'call': 'preadv2', 'reason': set([('fd', 'file')])},
14082               {'call': 'splice', 'reason': set([('fd', 'file')])},
14083               {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
14084               {'call': 'preadv', 'reason': set([('fd', 'file')])},
14085               {'call': 'getpeername', 'reason': set([('fd', 'file')])},
14086               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
14087               {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
14088               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
14089               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
14090               {'call': 'fcntl', 'reason': set([('fd', 'file')])},
14091               {'call': 'ioctl', 'reason': set([('fd', 'file')])},
14092               {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
14093               {'call': 'perf_event_open',
14094                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
14095               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
14096               {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
14097               {'call': 'futimesat', 'reason': set([('fd', 'file')])},
14098               {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
14099               {'call': 'shutdown', 'reason': set([('fd', 'file')])},
14100               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
14101               {'call': 'open', 'reason': set([('file', 'f_mode')])},
14102               {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
14103               {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
14104               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
14105               {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
14106               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
14107               {'call': 'getdents64', 'reason': set([('fd', 'file')])},
14108               {'call': 'listen', 'reason': set([('fd', 'file')])},
14109               {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
14110               {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
14111               {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
14112               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
14113               {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
14114               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
14115               {'call': 'fallocate', 'reason': set([('fd', 'file')])},
14116               {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
14117               {'call': 'llseek', 'reason': set([('fd', 'file')])},
```

```
14118                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
14119                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
14120                {'call': 'readv', 'reason': set([('fd', 'file')])},
14121                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
14122                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
14123                {'call': 'write', 'reason': set([('fd', 'file')])},
14124                {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
14125                {'call': 'sendfile', 'reason': set([('fd', 'file')])},
14126                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
14127                {'call': 'open_by_handle_at',
14128                 'reason': set([('file', 'f_mode')])},
14129                {'call': 'bind', 'reason': set([('fd', 'file')])},
14130                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
14131                {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
14132  'prlimit64': [{'call': 'keyctl',
14133                 'reason': set([('cred', 'user_ns'),
14134                                ('task_struct', 'cred'),
14135                                ('task_struct', 'group_leader'),
14136                                ('task_struct', 'real_cred'),
14137                                ('task_struct', 'sighand')])},
14138                {'call': 'rt_sigtimedwait',
14139                 'reason': set([('task_struct', 'cred'),
14140                                ('task_struct', 'group_leader'),
14141                                ('task_struct', 'real_cred'),
14142                                ('task_struct', 'sighand')])},
14143                {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
14144                {'call': 'msgrcv',
14145                 'reason': set([('task_struct', 'cred'),
14146                                ('task_struct', 'group_leader'),
14147                                ('task_struct', 'real_cred'),
14148                                ('task_struct', 'sighand')])},
14149                {'call': 'kill',
14150                 'reason': set([('task_struct', 'cred'),
14151                                ('task_struct', 'group_leader'),
14152                                ('task_struct', 'real_cred'),
14153                                ('task_struct', 'sighand')])},
14154                {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
14155                {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
14156                {'call': 'sched_getaffinity',
14157                 'reason': set([('task_struct', 'cred'),
14158                                ('task_struct', 'group_leader'),
14159                                ('task_struct', 'real_cred'),
14160                                ('task_struct', 'sighand')])},
14161                {'call': 'sched_setparam',
14162                 'reason': set([('task_struct', 'cred'),
14163                                ('task_struct', 'group_leader'),
14164                                ('task_struct', 'real_cred'),
14165                                ('task_struct', 'sighand')])},
14166                {'call': 'setgid',
14167                 'reason': set([('cred', 'egid'),
14168                                ('cred', 'gid'),
14169                                ('cred', 'sgid'),
14170                                ('cred', 'user_ns')])},
14171                {'call': 'ioprio_set',
14172                 'reason': set([('cred', 'user_ns'),
14173                                ('task_struct', 'cred'),
14174                                ('task_struct', 'group_leader'),
14175                                ('task_struct', 'real_cred'),
14176                                ('task_struct', 'sighand')])},
14177                {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
14178                {'call': 'getppid',
14179                 'reason': set([('task_struct', 'cred'),
14180                                ('task_struct', 'group_leader'),
14181                                ('task_struct', 'real_cred'),
14182                                ('task_struct', 'sighand')])},
14183                {'call': 'mq_timedreceive',
```



```
14184                 'reason': set([('task_struct', 'cred'),
14185                                ('task_struct', 'group_leader'),
14186                                ('task_struct', 'real_cred'),
14187                                ('task_struct', 'sighand')])},
14188                {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
14189                {'call': 'capget',
14190                 'reason': set([('task_struct', 'cred'),
14191                                ('task_struct', 'group_leader'),
14192                                ('task_struct', 'real_cred'),
14193                                ('task_struct', 'sighand')])},
14194                {'call': 'sched_setaffinity',
14195                 'reason': set([('cred', 'user_ns'),
14196                                ('task_struct', 'cred'),
14197                                ('task_struct', 'group_leader'),
14198                                ('task_struct', 'real_cred'),
14199                                ('task_struct', 'sighand')])},
14200                {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
14201                {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
14202                {'call': 'signal',
14203                 'reason': set([('task_struct', 'cred'),
14204                                ('task_struct', 'group_leader'),
14205                                ('task_struct', 'real_cred'),
14206                                ('task_struct', 'sighand')])},
14207                {'call': 'setreuid',
14208                 'reason': set([('cred', 'euid'),
14209                                ('cred', 'suid'),
14210                                ('cred', 'uid'),
14211                                ('cred', 'user_ns')])},
14212                {'call': 'semtimedop',
14213                 'reason': set([('task_struct', 'cred'),
14214                                ('task_struct', 'group_leader'),
14215                                ('task_struct', 'real_cred'),
14216                                ('task_struct', 'sighand')])},
14217                {'call': 'umount',
14218                 'reason': set([('task_struct', 'cred'),
14219                                ('task_struct', 'group_leader'),
14220                                ('task_struct', 'real_cred'),
14221                                ('task_struct', 'sighand')])},
14222                {'call': 'sched_rr_get_interval',
14223                 'reason': set([('task_struct', 'cred'),
14224                                ('task_struct', 'group_leader'),
14225                                ('task_struct', 'real_cred'),
14226                                ('task_struct', 'sighand')])},
14227                {'call': 'epoll_create1',
14228                 'reason': set([('cred', 'user_ns')])},
14229                {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
14230                {'call': 'rt_sigprocmask',
14231                 'reason': set([('task_struct', 'cred'),
14232                                ('task_struct', 'group_leader'),
14233                                ('task_struct', 'real_cred'),
14234                                ('task_struct', 'sighand')])},
14235                {'call': 'setsid',
14236                 'reason': set([('task_struct', 'cred'),
14237                                ('task_struct', 'group_leader'),
14238                                ('task_struct', 'real_cred'),
14239                                ('task_struct', 'sighand')])},
14240                {'call': 'sigaltstack',
14241                 'reason': set([('task_struct', 'cred'),
14242                                ('task_struct', 'group_leader'),
14243                                ('task_struct', 'real_cred'),
14244                                ('task_struct', 'sighand')])},
14245                {'call': 'sched_setattr',
14246                 'reason': set([('task_struct', 'cred'),
14247                                ('task_struct', 'group_leader'),
14248                                ('task_struct', 'real_cred'),
14249                                ('task_struct', 'sighand')])},
```

```
14250                {'call': 'setrlimit',
14251                 'reason': set([('rlimit', 'rlim_cur'),
14252                                ('rlimit', 'rlim_max')])},
14253                {'call': 'migrate_pages',
14254                 'reason': set([('cred', 'user_ns'),
14255                                ('task_struct', 'cred'),
14256                                ('task_struct', 'group_leader'),
14257                                ('task_struct', 'real_cred'),
14258                                ('task_struct', 'sighand')])},
14259                {'call': 'getitimer',
14260                 'reason': set([('task_struct', 'cred'),
14261                                ('task_struct', 'group_leader'),
14262                                ('task_struct', 'real_cred'),
14263                                ('task_struct', 'sighand')])},
14264                {'call': 'setpgid',
14265                 'reason': set([('task_struct', 'cred'),
14266                                ('task_struct', 'group_leader'),
14267                                ('task_struct', 'real_cred'),
14268                                ('task_struct', 'sighand')])},
14269                {'call': 'setresgid',
14270                 'reason': set([('cred', 'egid'),
14271                                ('cred', 'gid'),
14272                                ('cred', 'sgid'),
14273                                ('cred', 'user_ns')])},
14274                {'call': 'setregid',
14275                 'reason': set([('cred', 'egid'),
14276                                ('cred', 'gid'),
14277                                ('cred', 'sgid'),
14278                                ('cred', 'user_ns')])},
14279                {'call': 'getsid',
14280                 'reason': set([('task_struct', 'cred'),
14281                                ('task_struct', 'group_leader'),
14282                                ('task_struct', 'real_cred'),
14283                                ('task_struct', 'sighand')])},
14284                {'call': 'old_getrlimit',
14285                 'reason': set([('rlimit', 'rlim_cur'),
14286                                ('rlimit', 'rlim_max')])},
14287                {'call': 'prlimit64',
14288                 'reason': set([('cred', 'user_ns'),
14289                                ('rlimit', 'rlim_cur'),
14290                                ('rlimit', 'rlim_max'),
14291                                ('rlimit64', 'rlim_cur'),
14292                                ('rlimit64', 'rlim_max'),
14293                                ('task_struct', 'cred'),
14294                                ('task_struct', 'group_leader'),
14295                                ('task_struct', 'real_cred'),
14296                                ('task_struct', 'sighand')])},
14297                {'call': 'perf_event_open',
14298                 'reason': set([('task_struct', 'cred'),
14299                                ('task_struct', 'group_leader'),
14300                                ('task_struct', 'real_cred'),
14301                                ('task_struct', 'sighand')])},
14302                {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
14303                {'call': 'rt_sigaction',
14304                 'reason': set([('task_struct', 'cred'),
14305                                ('task_struct', 'group_leader'),
14306                                ('task_struct', 'real_cred'),
14307                                ('task_struct', 'sighand')])},
14308                {'call': 'getpgid',
14309                 'reason': set([('task_struct', 'cred'),
14310                                ('task_struct', 'group_leader'),
14311                                ('task_struct', 'real_cred'),
14312                                ('task_struct', 'sighand')])},
14313                {'call': 'getpriority',
14314                 'reason': set([('cred', 'user_ns'),
14315                                ('task_struct', 'cred'),
```



```
14316                                ('task_struct', 'group_leader'),
14317                                ('task_struct', 'real_cred'),
14318                                ('task_struct', 'sighand')])},
14319                {'call': 'sigaction',
14320                 'reason': set([('task_struct', 'cred'),
14321                                ('task_struct', 'group_leader'),
14322                                ('task_struct', 'real_cred'),
14323                                ('task_struct', 'sighand')])},
14324                {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
14325                {'call': 'setns',
14326                 'reason': set([('task_struct', 'cred'),
14327                                ('task_struct', 'group_leader'),
14328                                ('task_struct', 'real_cred'),
14329                                ('task_struct', 'sighand')])},
14330                {'call': 'fork',
14331                 'reason': set([('task_struct', 'cred'),
14332                                ('task_struct', 'group_leader'),
14333                                ('task_struct', 'real_cred'),
14334                                ('task_struct', 'sighand')])},
14335                {'call': 'get_robust_list',
14336                 'reason': set([('task_struct', 'cred'),
14337                                ('task_struct', 'group_leader'),
14338                                ('task_struct', 'real_cred'),
14339                                ('task_struct', 'sighand')])},
14340                {'call': 'mq_timedsend',
14341                 'reason': set([('task_struct', 'cred'),
14342                                ('task_struct', 'group_leader'),
14343                                ('task_struct', 'real_cred'),
14344                                ('task_struct', 'sighand')])},
14345                {'call': 'sched_getscheduler',
14346                 'reason': set([('task_struct', 'cred'),
14347                                ('task_struct', 'group_leader'),
14348                                ('task_struct', 'real_cred'),
14349                                ('task_struct', 'sighand')])},
14350                {'call': 'ptrace',
14351                 'reason': set([('task_struct', 'cred'),
14352                                ('task_struct', 'group_leader'),
14353                                ('task_struct', 'real_cred'),
14354                                ('task_struct', 'sighand')])},
14355                {'call': 'sched_getattr',
14356                 'reason': set([('task_struct', 'cred'),
14357                                ('task_struct', 'group_leader'),
14358                                ('task_struct', 'real_cred'),
14359                                ('task_struct', 'sighand')])},
14360                {'call': 'getrusage',
14361                 'reason': set([('task_struct', 'cred'),
14362                                ('task_struct', 'group_leader'),
14363                                ('task_struct', 'real_cred'),
14364                                ('task_struct', 'sighand')])},
14365                {'call': 'sched_setscheduler',
14366                 'reason': set([('task_struct', 'cred'),
14367                                ('task_struct', 'group_leader'),
14368                                ('task_struct', 'real_cred'),
14369                                ('task_struct', 'sighand')])},
14370                {'call': 'setresuid',
14371                 'reason': set([('cred', 'euid'),
14372                                ('cred', 'suid'),
14373                                ('cred', 'uid'),
14374                                ('cred', 'user_ns')])},
14375                {'call': 'setitimer',
14376                 'reason': set([('task_struct', 'cred'),
14377                                ('task_struct', 'group_leader'),
14378                                ('task_struct', 'real_cred'),
14379                                ('task_struct', 'sighand')])},
14380                {'call': 'ioprio_get',
14381                 'reason': set([('cred', 'user_ns'),
```

```
                                ('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'vfork',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'setuid',
               'reason': set([('cred', 'euid'),
                                ('cred', 'suid'),
                                ('cred', 'uid'),
                                ('cred', 'user_ns')])},
              {'call': 'prctl',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'move_pages',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
              {'call': 'setpriority',
               'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'clone',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'sched_getparam',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])}],
 'pselect6': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('mm_segment_t', 'seg'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_unlink',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'swapoff',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'sched_getaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sched_setparam',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchmod',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'memfd_create',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'ioprio_set',
               'reason': set([('mm_segment_t', 'seg')])},
```



```
              {'call': 'readlinkat',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'io_getevents',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchown',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_timedreceive',
               'reason': set([('mm_segment_t', 'seg'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'utime',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'sched_setaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'semtimedop',
               'reason': set([('mm_segment_t', 'seg'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'settimeofday',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('mm_segment_t', 'seg'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'timerfd_gettime',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'pselect6',
               'reason': set([('compat_timespec', 'tv_nsec'),
                                ('compat_timespec', 'tv_sec'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'uselib',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sigaltstack',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sched_setattr',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'migrate_pages',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchmodat',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'inotify_add_watch',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'timer_settime',
               'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
              {'call': 'ftruncate',
```

```
14514                 'reason': set([('timespec', 'tv_nsec'),
14515                                ('timespec', 'tv_sec')])},
14516               {'call': 'timer_gettime',
14517                'reason': set([('timespec', 'tv_nsec'),
14518                               ('timespec', 'tv_sec')])},
14519               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
14520               {'call': 'ioctl',
14521                'reason': set([('timespec', 'tv_nsec'),
14522                               ('timespec', 'tv_sec')])},
14523               {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
14524               {'call': 'perf_event_open',
14525                'reason': set([('mm_segment_t', 'seg')])},
14526               {'call': 'linkat',
14527                'reason': set([('timespec', 'tv_nsec'),
14528                               ('timespec', 'tv_sec')])},
14529               {'call': 'stime',
14530                'reason': set([('timespec', 'tv_nsec'),
14531                               ('timespec', 'tv_sec')])},
14532               {'call': 'rt_sigaction',
14533                'reason': set([('mm_segment_t', 'seg')])},
14534               {'call': 'futimesat',
14535                'reason': set([('timespec', 'tv_nsec'),
14536                               ('timespec', 'tv_sec')])},
14537               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
14538               {'call': 'poll',
14539                'reason': set([('timespec', 'tv_nsec'),
14540                               ('timespec', 'tv_sec')])},
14541               {'call': 'select',
14542                'reason': set([('compat_timespec', 'tv_nsec'),
14543                               ('compat_timespec', 'tv_sec'),
14544                               ('timespec', 'tv_nsec'),
14545                               ('timespec', 'tv_sec')])},
14546               {'call': 'unlink',
14547                'reason': set([('timespec', 'tv_nsec'),
14548                               ('timespec', 'tv_sec')])},
14549               {'call': 'getpriority',
14550                'reason': set([('mm_segment_t', 'seg')])},
14551               {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
14552               {'call': 'nanosleep',
14553                'reason': set([('compat_timespec', 'tv_nsec'),
14554                               ('compat_timespec', 'tv_sec'),
14555                               ('timespec', 'tv_nsec'),
14556                               ('timespec', 'tv_sec')])},
14557               {'call': 'mq_getsetattr',
14558                'reason': set([('timespec', 'tv_nsec'),
14559                               ('timespec', 'tv_sec')])},
14560               {'call': 'faccessat',
14561                'reason': set([('timespec', 'tv_nsec'),
14562                               ('timespec', 'tv_sec')])},
14563               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
14564               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
14565               {'call': 'get_robust_list',
14566                'reason': set([('mm_segment_t', 'seg')])},
14567               {'call': 'mq_timedsend',
14568                'reason': set([('mm_segment_t', 'seg'),
14569                               ('timespec', 'tv_nsec'),
14570                               ('timespec', 'tv_sec')])},
14571               {'call': 'sched_getscheduler',
14572                'reason': set([('mm_segment_t', 'seg')])},
14573               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
14574               {'call': 'swapon',
14575                'reason': set([('timespec', 'tv_nsec'),
14576                               ('timespec', 'tv_sec')])},
14577               {'call': 'epoll_wait',
14578                'reason': set([('timespec', 'tv_nsec'),
14579                               ('timespec', 'tv_sec')])},
```



```
14580               {'call': 'sched_getattr',
14581                'reason': set([('mm_segment_t', 'seg')])},
14582               {'call': 'fchownat',
14583                'reason': set([('timespec', 'tv_nsec'),
14584                               ('timespec', 'tv_sec')])},
14585               {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
14586               {'call': 'fstat',
14587                'reason': set([('timespec', 'tv_nsec'),
14588                               ('timespec', 'tv_sec')])},
14589               {'call': 'timerfd_settime',
14590                'reason': set([('timespec', 'tv_nsec'),
14591                               ('timespec', 'tv_sec')])},
14592               {'call': 'sched_setscheduler',
14593                'reason': set([('mm_segment_t', 'seg')])},
14594               {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
14595               {'call': 'ioprio_get',
14596                'reason': set([('mm_segment_t', 'seg')])},
14597               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
14598               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
14599               {'call': 'move_pages',
14600                'reason': set([('mm_segment_t', 'seg')])},
14601               {'call': 'setpriority',
14602                'reason': set([('mm_segment_t', 'seg')])},
14603               {'call': 'mq_notify',
14604                'reason': set([('timespec', 'tv_nsec'),
14605                               ('timespec', 'tv_sec')])},
14606               {'call': 'sendfile',
14607                'reason': set([('timespec', 'tv_nsec'),
14608                               ('timespec', 'tv_sec')])},
14609               {'call': 'newfstat',
14610                'reason': set([('timespec', 'tv_nsec'),
14611                               ('timespec', 'tv_sec')])},
14612               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
14613               {'call': 'clock_nanosleep',
14614                'reason': set([('compat_timespec', 'tv_nsec'),
14615                               ('compat_timespec', 'tv_sec'),
14616                               ('timespec', 'tv_nsec'),
14617                               ('timespec', 'tv_sec')])},
14618               {'call': 'unlinkat',
14619                'reason': set([('timespec', 'tv_nsec'),
14620                               ('timespec', 'tv_sec')])},
14621               {'call': 'sched_getparam',
14622                'reason': set([('mm_segment_t', 'seg')])},
14623               {'call': 'futex',
14624                'reason': set([('timespec', 'tv_nsec'),
14625                               ('timespec', 'tv_sec')])},
14626               {'call': 'recvmmsg',
14627                'reason': set([('timespec', 'tv_nsec'),
14628                               ('timespec', 'tv_sec')])},
14629               {'call': 'sendfile64',
14630                'reason': set([('timespec', 'tv_nsec'),
14631                               ('timespec', 'tv_sec')])},
14632               {'call': 'ppoll',
14633                'reason': set([('compat_timespec', 'tv_nsec'),
14634                               ('compat_timespec', 'tv_sec'),
14635                               ('timespec', 'tv_nsec'),
14636                               ('timespec', 'tv_sec')])}],
14637  'ptrace': [{'call': 'keyctl',
14638              'reason': set([('task_struct', 'exit_state'),
14639                             ('task_struct', 'flags'),
14640                             ('task_struct', 'parent'),
14641                             ('task_struct', 'ptrace'),
14642                             ('task_struct', 'real_parent'),
14643                             ('task_struct', 'state')])},
14644             {'call': 'rt_sigtimedwait',
14645              'reason': set([('task_struct', 'exit_state'),
```

```
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'msgrcv',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'kill',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'pause', 'reason': set([('task_struct', 'state')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'ioprio_set',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'getppid',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'capget',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_setaffinity',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
```



```
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'signal',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'semtimedop',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'umount',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'setsid',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sigaltstack',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
```

```
                                ('task_struct', 'state')])},
            {'call': 'getitimer',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'setpgid',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'rt_sigsuspend',
             'reason': set([('task_struct', 'state')])},
            {'call': 'getsid',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'prlimit64',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'perf_event_open',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'getpgid',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'getpriority',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sigaction',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
```



```
            {'call': 'setns',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'fork',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'ptrace',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'epoll_wait', 'reason': set([('task_struct', 'state')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'getrusage',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setitimer',
```

```
14910              'reason': set([('task_struct', 'exit_state'),
14911                             ('task_struct', 'flags'),
14912                             ('task_struct', 'parent'),
14913                             ('task_struct', 'ptrace'),
14914                             ('task_struct', 'real_parent'),
14915                             ('task_struct', 'state')])},
14916            {'call': 'ioprio_get',
14917             'reason': set([('task_struct', 'exit_state'),
14918                             ('task_struct', 'flags'),
14919                             ('task_struct', 'parent'),
14920                             ('task_struct', 'ptrace'),
14921                             ('task_struct', 'real_parent'),
14922                             ('task_struct', 'state')])},
14923            {'call': 'vfork',
14924             'reason': set([('task_struct', 'exit_state'),
14925                             ('task_struct', 'flags'),
14926                             ('task_struct', 'parent'),
14927                             ('task_struct', 'ptrace'),
14928                             ('task_struct', 'real_parent'),
14929                             ('task_struct', 'state')])},
14930            {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
14931            {'call': 'prctl',
14932             'reason': set([('task_struct', 'exit_state'),
14933                             ('task_struct', 'flags'),
14934                             ('task_struct', 'parent'),
14935                             ('task_struct', 'ptrace'),
14936                             ('task_struct', 'real_parent'),
14937                             ('task_struct', 'state')])},
14938            {'call': 'move_pages',
14939             'reason': set([('task_struct', 'exit_state'),
14940                             ('task_struct', 'flags'),
14941                             ('task_struct', 'parent'),
14942                             ('task_struct', 'ptrace'),
14943                             ('task_struct', 'real_parent'),
14944                             ('task_struct', 'state')])},
14945            {'call': 'setpriority',
14946             'reason': set([('task_struct', 'exit_state'),
14947                             ('task_struct', 'flags'),
14948                             ('task_struct', 'parent'),
14949                             ('task_struct', 'ptrace'),
14950                             ('task_struct', 'real_parent'),
14951                             ('task_struct', 'state')])},
14952            {'call': 'clone',
14953             'reason': set([('task_struct', 'exit_state'),
14954                             ('task_struct', 'flags'),
14955                             ('task_struct', 'parent'),
14956                             ('task_struct', 'ptrace'),
14957                             ('task_struct', 'real_parent'),
14958                             ('task_struct', 'state')])},
14959            {'call': 'sigsuspend', 'reason': set([('task_struct', 'state')])},
14960            {'call': 'sched_getparam',
14961             'reason': set([('task_struct', 'exit_state'),
14962                             ('task_struct', 'flags'),
14963                             ('task_struct', 'parent'),
14964                             ('task_struct', 'ptrace'),
14965                             ('task_struct', 'real_parent'),
14966                             ('task_struct', 'state')])}],
14967  'pwrite64': [{'call': 'syncfs',
14968                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14969               {'call': 'vmsplice',
14970                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14971               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
14972               {'call': 'pwritev64',
14973                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14974               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
14975               {'call': 'fremovexattr',
```



```
14976                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14977               {'call': 'readahead',
14978                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14979               {'call': 'getdents',
14980                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14981               {'call': 'writev',
14982                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14983               {'call': 'preadv64',
14984                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14985               {'call': 'fchmod',
14986                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14987               {'call': 'pread64',
14988                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14989               {'call': 'signalfd4',
14990                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14991               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
14992               {'call': 'remap_file_pages',
14993                'reason': set([('file', 'f_mode')])},
14994               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
14995               {'call': 'read',
14996                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14997               {'call': 'fchown',
14998                'reason': set([('fd', 'file'), ('fd', 'flags')])},
14999               {'call': 'mq_timedreceive',
15000                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15001               {'call': 'utime',
15002                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15003               {'call': 'fsync',
15004                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15005               {'call': 'bpf',
15006                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15007               {'call': 'recvfrom',
15008                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15009               {'call': 'fsetxattr',
15010                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15011               {'call': 'sendto',
15012                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15013               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
15014               {'call': 'tee',
15015                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15016               {'call': 'sync_file_range',
15017                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15018               {'call': 'lseek',
15019                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15020               {'call': 'connect',
15021                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15022               {'call': 'getsockname',
15023                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15024               {'call': 'epoll_ctl',
15025                'reason': set([('fd', 'file'),
15026                               ('fd', 'flags'),
15027                               ('file', 'f_mode')])},
15028               {'call': 'flock',
15029                'reason': set([('fd', 'file'),
15030                               ('fd', 'flags'),
15031                               ('file', 'f_mode')])},
15032               {'call': 'pwritev',
15033                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15034               {'call': 'fchdir',
15035                'reason': set([('fd', 'file'), ('fd', 'flags')])},
15036               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
15037               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
15038               {'call': 'accept4',
15039                'reason': set([('fd', 'file'),
15040                               ('fd', 'flags'),
15041                               ('file', 'f_mode')])},
```

```
15042                {'call': 'old_readdir',
15043                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15044                {'call': 'inotify_rm_watch',
15045                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15046                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
15047                {'call': 'utimensat',
15048                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15049                {'call': 'inotify_add_watch',
15050                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15051                {'call': 'preadv2',
15052                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15053                {'call': 'splice',
15054                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15055                {'call': 'ftruncate',
15056                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15057                {'call': 'preadv',
15058                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15059                {'call': 'getpeername',
15060                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15061                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
15062                {'call': 'setsockopt',
15063                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15064                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
15065                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
15066                {'call': 'fcntl',
15067                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15068                {'call': 'ioctl',
15069                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15070                {'call': 'pwrite64',
15071                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15072                {'call': 'perf_event_open',
15073                 'reason': set([('fd', 'file'),
15074                                ('fd', 'flags'),
15075                                ('file', 'f_mode')])},
15076                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
15077                {'call': 'pwritev64v2',
15078                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15079                {'call': 'futimesat',
15080                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15081                {'call': 'pwritev2',
15082                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15083                {'call': 'shutdown',
15084                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15085                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
15086                {'call': 'open', 'reason': set([('file', 'f_mode')])},
15087                {'call': 'getsockopt',
15088                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15089                {'call': 'mq_getsetattr',
15090                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15091                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
15092                {'call': 'fdatasync',
15093                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15094                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
15095                {'call': 'getdents64',
15096                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15097                {'call': 'listen',
15098                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15099                {'call': 'copy_file_range',
15100                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15101                {'call': 'mq_timedsend',
15102                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15103                {'call': 'fgetxattr',
15104                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15105                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
15106                {'call': 'fcntl64',
15107                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
15108                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
15109                {'call': 'fallocate',
15110                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15111                {'call': 'epoll_wait',
15112                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15113                {'call': 'llseek',
15114                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15115                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
15116                {'call': 'preadv64v2',
15117                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15118                {'call': 'readv',
15119                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15120                {'call': 'fstatfs',
15121                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15122                {'call': 'fstatfs64',
15123                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15124                {'call': 'write',
15125                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15126                {'call': 'mq_notify',
15127                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15128                {'call': 'sendfile',
15129                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15130                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
15131                {'call': 'open_by_handle_at',
15132                 'reason': set([('file', 'f_mode')])},
15133                {'call': 'bind',
15134                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15135                {'call': 'flistxattr',
15136                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15137                {'call': 'sendfile64',
15138                 'reason': set([('fd', 'file'), ('fd', 'flags')])}],
15139 'pwritev': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
15140             {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
15141             {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
15142             {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
15143             {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
15144             {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
15145             {'call': 'readahead', 'reason': set([('fd', 'file')])},
15146             {'call': 'getdents', 'reason': set([('fd', 'file')])},
15147             {'call': 'writev', 'reason': set([('fd', 'file')])},
15148             {'call': 'preadv64', 'reason': set([('fd', 'file')])},
15149             {'call': 'fchmod', 'reason': set([('fd', 'file')])},
15150             {'call': 'pread64', 'reason': set([('fd', 'file')])},
15151             {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
15152             {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
15153             {'call': 'remap_file_pages',
15154              'reason': set([('file', 'f_mode')])},
15155             {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
15156             {'call': 'read', 'reason': set([('fd', 'file')])},
15157             {'call': 'fchown', 'reason': set([('fd', 'file')])},
15158             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
15159             {'call': 'utime', 'reason': set([('fd', 'file')])},
15160             {'call': 'fsync', 'reason': set([('fd', 'file')])},
15161             {'call': 'bpf', 'reason': set([('fd', 'file')])},
15162             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
15163             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
15164             {'call': 'sendto', 'reason': set([('fd', 'file')])},
15165             {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
15166             {'call': 'tee', 'reason': set([('fd', 'file')])},
15167             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
15168             {'call': 'lseek', 'reason': set([('fd', 'file')])},
15169             {'call': 'connect', 'reason': set([('fd', 'file')])},
15170             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
15171             {'call': 'epoll_ctl',
15172              'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15173             {'call': 'flock',
```

```
15174                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15175               {'call': 'pwritev', 'reason': set([('fd', 'file')])},
15176               {'call': 'fchdir', 'reason': set([('fd', 'file')])},
15177               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
15178               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
15179               {'call': 'accept4',
15180                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15181               {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
15182               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
15183               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
15184               {'call': 'utimensat', 'reason': set([('fd', 'file')])},
15185               {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
15186               {'call': 'preadv2', 'reason': set([('fd', 'file')])},
15187               {'call': 'splice', 'reason': set([('fd', 'file')])},
15188               {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
15189               {'call': 'preadv', 'reason': set([('fd', 'file')])},
15190               {'call': 'getpeername', 'reason': set([('fd', 'file')])},
15191               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
15192               {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
15193               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
15194               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
15195               {'call': 'fcntl', 'reason': set([('fd', 'file')])},
15196               {'call': 'ioctl', 'reason': set([('fd', 'file')])},
15197               {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
15198               {'call': 'perf_event_open',
15199                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15200               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
15201               {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
15202               {'call': 'futimesat', 'reason': set([('fd', 'file')])},
15203               {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
15204               {'call': 'shutdown', 'reason': set([('fd', 'file')])},
15205               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
15206               {'call': 'open', 'reason': set([('file', 'f_mode')])},
15207               {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
15208               {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
15209               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
15210               {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
15211               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
15212               {'call': 'getdents64', 'reason': set([('fd', 'file')])},
15213               {'call': 'listen', 'reason': set([('fd', 'file')])},
15214               {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
15215               {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
15216               {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
15217               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
15218               {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
15219               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
15220               {'call': 'fallocate', 'reason': set([('fd', 'file')])},
15221               {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
15222               {'call': 'llseek', 'reason': set([('fd', 'file')])},
15223               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
15224               {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
15225               {'call': 'readv', 'reason': set([('fd', 'file')])},
15226               {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
15227               {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
15228               {'call': 'write', 'reason': set([('fd', 'file')])},
15229               {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
15230               {'call': 'sendfile', 'reason': set([('fd', 'file')])},
15231               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
15232               {'call': 'open_by_handle_at',
15233                'reason': set([('file', 'f_mode')])},
15234               {'call': 'bind', 'reason': set([('fd', 'file')])},
15235               {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
15236               {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
15237  'pwritev2': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
15238               {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
15239               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
```



```
15240               {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
15241               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
15242               {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
15243               {'call': 'readahead', 'reason': set([('fd', 'file')])},
15244               {'call': 'getdents', 'reason': set([('fd', 'file')])},
15245               {'call': 'writev', 'reason': set([('fd', 'file')])},
15246               {'call': 'preadv64', 'reason': set([('fd', 'file')])},
15247               {'call': 'fchmod', 'reason': set([('fd', 'file')])},
15248               {'call': 'pread64', 'reason': set([('fd', 'file')])},
15249               {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
15250               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
15251               {'call': 'remap_file_pages',
15252                'reason': set([('file', 'f_mode')])},
15253               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
15254               {'call': 'read', 'reason': set([('fd', 'file')])},
15255               {'call': 'fchown', 'reason': set([('fd', 'file')])},
15256               {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
15257               {'call': 'utime', 'reason': set([('fd', 'file')])},
15258               {'call': 'fsync', 'reason': set([('fd', 'file')])},
15259               {'call': 'bpf', 'reason': set([('fd', 'file')])},
15260               {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
15261               {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
15262               {'call': 'sendto', 'reason': set([('fd', 'file')])},
15263               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
15264               {'call': 'tee', 'reason': set([('fd', 'file')])},
15265               {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
15266               {'call': 'lseek', 'reason': set([('fd', 'file')])},
15267               {'call': 'connect', 'reason': set([('fd', 'file')])},
15268               {'call': 'getsockname', 'reason': set([('fd', 'file')])},
15269               {'call': 'epoll_ctl',
15270                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15271               {'call': 'flock',
15272                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15273               {'call': 'pwritev', 'reason': set([('fd', 'file')])},
15274               {'call': 'fchdir', 'reason': set([('fd', 'file')])},
15275               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
15276               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
15277               {'call': 'accept4',
15278                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15279               {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
15280               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
15281               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
15282               {'call': 'utimensat', 'reason': set([('fd', 'file')])},
15283               {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
15284               {'call': 'preadv2', 'reason': set([('fd', 'file')])},
15285               {'call': 'splice', 'reason': set([('fd', 'file')])},
15286               {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
15287               {'call': 'preadv', 'reason': set([('fd', 'file')])},
15288               {'call': 'getpeername', 'reason': set([('fd', 'file')])},
15289               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
15290               {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
15291               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
15292               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
15293               {'call': 'fcntl', 'reason': set([('fd', 'file')])},
15294               {'call': 'ioctl', 'reason': set([('fd', 'file')])},
15295               {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
15296               {'call': 'perf_event_open',
15297                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15298               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
15299               {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
15300               {'call': 'futimesat', 'reason': set([('fd', 'file')])},
15301               {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
15302               {'call': 'shutdown', 'reason': set([('fd', 'file')])},
15303               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
15304               {'call': 'open', 'reason': set([('file', 'f_mode')])},
15305               {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
```

```
15306               {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
15307               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
15308               {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
15309               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
15310               {'call': 'getdents64', 'reason': set([('fd', 'file')])},
15311               {'call': 'listen', 'reason': set([('fd', 'file')])},
15312               {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
15313               {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
15314               {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
15315               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
15316               {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
15317               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
15318               {'call': 'fallocate', 'reason': set([('fd', 'file')])},
15319               {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
15320               {'call': 'llseek', 'reason': set([('fd', 'file')])},
15321               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
15322               {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
15323               {'call': 'readv', 'reason': set([('fd', 'file')])},
15324               {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
15325               {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
15326               {'call': 'write', 'reason': set([('fd', 'file')])},
15327               {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
15328               {'call': 'sendfile', 'reason': set([('fd', 'file')])},
15329               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
15330               {'call': 'open_by_handle_at',
15331                'reason': set([('file', 'f_mode')])},
15332               {'call': 'bind', 'reason': set([('fd', 'file')])},
15333               {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
15334               {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
15335  'pwritev64': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
15336                {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
15337                {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
15338                {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
15339                {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
15340                {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
15341                {'call': 'readahead', 'reason': set([('fd', 'file')])},
15342                {'call': 'getdents', 'reason': set([('fd', 'file')])},
15343                {'call': 'writev', 'reason': set([('fd', 'file')])},
15344                {'call': 'preadv64', 'reason': set([('fd', 'file')])},
15345                {'call': 'fchmod', 'reason': set([('fd', 'file')])},
15346                {'call': 'pread64', 'reason': set([('fd', 'file')])},
15347                {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
15348                {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
15349                {'call': 'remap_file_pages',
15350                 'reason': set([('file', 'f_mode')])},
15351                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
15352                {'call': 'read', 'reason': set([('fd', 'file')])},
15353                {'call': 'fchown', 'reason': set([('fd', 'file')])},
15354                {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
15355                {'call': 'utime', 'reason': set([('fd', 'file')])},
15356                {'call': 'fsync', 'reason': set([('fd', 'file')])},
15357                {'call': 'bpf', 'reason': set([('fd', 'file')])},
15358                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
15359                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
15360                {'call': 'sendto', 'reason': set([('fd', 'file')])},
15361                {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
15362                {'call': 'tee', 'reason': set([('fd', 'file')])},
15363                {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
15364                {'call': 'lseek', 'reason': set([('fd', 'file')])},
15365                {'call': 'connect', 'reason': set([('fd', 'file')])},
15366                {'call': 'getsockname', 'reason': set([('fd', 'file')])},
15367                {'call': 'epoll_ctl',
15368                 'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15369                {'call': 'flock',
15370                 'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15371                {'call': 'pwritev', 'reason': set([('fd', 'file')])},
```



```
15372               {'call': 'fchdir', 'reason': set([('fd', 'file')])},
15373               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
15374               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
15375               {'call': 'accept4',
15376                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15377               {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
15378               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
15379               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
15380               {'call': 'utimensat', 'reason': set([('fd', 'file')])},
15381               {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
15382               {'call': 'preadv2', 'reason': set([('fd', 'file')])},
15383               {'call': 'splice', 'reason': set([('fd', 'file')])},
15384               {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
15385               {'call': 'preadv', 'reason': set([('fd', 'file')])},
15386               {'call': 'getpeername', 'reason': set([('fd', 'file')])},
15387               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
15388               {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
15389               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
15390               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
15391               {'call': 'fcntl', 'reason': set([('fd', 'file')])},
15392               {'call': 'ioctl', 'reason': set([('fd', 'file')])},
15393               {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
15394               {'call': 'perf_event_open',
15395                'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15396               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
15397               {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
15398               {'call': 'futimesat', 'reason': set([('fd', 'file')])},
15399               {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
15400               {'call': 'shutdown', 'reason': set([('fd', 'file')])},
15401               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
15402               {'call': 'open', 'reason': set([('file', 'f_mode')])},
15403               {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
15404               {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
15405               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
15406               {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
15407               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
15408               {'call': 'getdents64', 'reason': set([('fd', 'file')])},
15409               {'call': 'listen', 'reason': set([('fd', 'file')])},
15410               {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
15411               {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
15412               {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
15413               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
15414               {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
15415               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
15416               {'call': 'fallocate', 'reason': set([('fd', 'file')])},
15417               {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
15418               {'call': 'llseek', 'reason': set([('fd', 'file')])},
15419               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
15420               {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
15421               {'call': 'readv', 'reason': set([('fd', 'file')])},
15422               {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
15423               {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
15424               {'call': 'write', 'reason': set([('fd', 'file')])},
15425               {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
15426               {'call': 'sendfile', 'reason': set([('fd', 'file')])},
15427               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
15428               {'call': 'open_by_handle_at',
15429                'reason': set([('file', 'f_mode')])},
15430               {'call': 'bind', 'reason': set([('fd', 'file')])},
15431               {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
15432               {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
15433  'pwritev64v2': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
15434                  {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
15435                  {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
15436                  {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
15437                  {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
```

```
15438                 {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
15439                 {'call': 'readahead', 'reason': set([('fd', 'file')])},
15440                 {'call': 'getdents', 'reason': set([('fd', 'file')])},
15441                 {'call': 'writev', 'reason': set([('fd', 'file')])},
15442                 {'call': 'preadv64', 'reason': set([('fd', 'file')])},
15443                 {'call': 'fchmod', 'reason': set([('fd', 'file')])},
15444                 {'call': 'pread64', 'reason': set([('fd', 'file')])},
15445                 {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
15446                 {'call': 'memfd_create',
15447                  'reason': set([('file', 'f_mode')])},
15448                 {'call': 'remap_file_pages',
15449                  'reason': set([('file', 'f_mode')])},
15450                 {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
15451                 {'call': 'read', 'reason': set([('fd', 'file')])},
15452                 {'call': 'fchown', 'reason': set([('fd', 'file')])},
15453                 {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
15454                 {'call': 'utime', 'reason': set([('fd', 'file')])},
15455                 {'call': 'fsync', 'reason': set([('fd', 'file')])},
15456                 {'call': 'bpf', 'reason': set([('fd', 'file')])},
15457                 {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
15458                 {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
15459                 {'call': 'sendto', 'reason': set([('fd', 'file')])},
15460                 {'call': 'epoll_create1',
15461                  'reason': set([('file', 'f_mode')])},
15462                 {'call': 'tee', 'reason': set([('fd', 'file')])},
15463                 {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
15464                 {'call': 'lseek', 'reason': set([('fd', 'file')])},
15465                 {'call': 'connect', 'reason': set([('fd', 'file')])},
15466                 {'call': 'getsockname', 'reason': set([('fd', 'file')])},
15467                 {'call': 'epoll_ctl',
15468                  'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15469                 {'call': 'flock',
15470                  'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15471                 {'call': 'pwritev', 'reason': set([('fd', 'file')])},
15472                 {'call': 'fchdir', 'reason': set([('fd', 'file')])},
15473                 {'call': 'openat', 'reason': set([('file', 'f_mode')])},
15474                 {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
15475                 {'call': 'accept4',
15476                  'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15477                 {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
15478                 {'call': 'inotify_rm_watch',
15479                  'reason': set([('fd', 'file')])},
15480                 {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
15481                 {'call': 'utimensat', 'reason': set([('fd', 'file')])},
15482                 {'call': 'inotify_add_watch',
15483                  'reason': set([('fd', 'file')])},
15484                 {'call': 'preadv2', 'reason': set([('fd', 'file')])},
15485                 {'call': 'splice', 'reason': set([('fd', 'file')])},
15486                 {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
15487                 {'call': 'preadv', 'reason': set([('fd', 'file')])},
15488                 {'call': 'getpeername', 'reason': set([('fd', 'file')])},
15489                 {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
15490                 {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
15491                 {'call': 'socket', 'reason': set([('file', 'f_mode')])},
15492                 {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
15493                 {'call': 'fcntl', 'reason': set([('fd', 'file')])},
15494                 {'call': 'ioctl', 'reason': set([('fd', 'file')])},
15495                 {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
15496                 {'call': 'perf_event_open',
15497                  'reason': set([('fd', 'file'), ('file', 'f_mode')])},
15498                 {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
15499                 {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
15500                 {'call': 'futimesat', 'reason': set([('fd', 'file')])},
15501                 {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
15502                 {'call': 'shutdown', 'reason': set([('fd', 'file')])},
15503                 {'call': 'acct', 'reason': set([('file', 'f_mode')])},
```



```
15504                 {'call': 'open', 'reason': set([('file', 'f_mode')])},
15505                 {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
15506                 {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
15507                 {'call': 'dup', 'reason': set([('file', 'f_mode')])},
15508                 {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
15509                 {'call': 'setns', 'reason': set([('file', 'f_mode')])},
15510                 {'call': 'getdents64', 'reason': set([('fd', 'file')])},
15511                 {'call': 'listen', 'reason': set([('fd', 'file')])},
15512                 {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
15513                 {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
15514                 {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
15515                 {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
15516                 {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
15517                 {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
15518                 {'call': 'fallocate', 'reason': set([('fd', 'file')])},
15519                 {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
15520                 {'call': 'llseek', 'reason': set([('fd', 'file')])},
15521                 {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
15522                 {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
15523                 {'call': 'readv', 'reason': set([('fd', 'file')])},
15524                 {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
15525                 {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
15526                 {'call': 'write', 'reason': set([('fd', 'file')])},
15527                 {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
15528                 {'call': 'sendfile', 'reason': set([('fd', 'file')])},
15529                 {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
15530                 {'call': 'open_by_handle_at',
15531                  'reason': set([('file', 'f_mode')])},
15532                 {'call': 'bind', 'reason': set([('fd', 'file')])},
15533                 {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
15534                 {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
15535  'quotactl': [{'call': 'syncfs',
15536                 'reason': set([('super_block', 's_flags'),
15537                                ('super_block', 's_qcop'),
15538                                ('super_block', 's_quota_types')])},
15539                {'call': 'sysfs', 'reason': set([('filename', 'name')])},
15540                {'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
15541                {'call': 'swapoff', 'reason': set([('filename', 'name')])},
15542                {'call': 'ustat',
15543                 'reason': set([('super_block', 's_flags'),
15544                                ('super_block', 's_qcop'),
15545                                ('super_block', 's_quota_types')])},
15546                {'call': 'umount',
15547                 'reason': set([('super_block', 's_flags'),
15548                                ('super_block', 's_qcop'),
15549                                ('super_block', 's_quota_types')])},
15550                {'call': 'openat', 'reason': set([('filename', 'name')])},
15551                {'call': 'uselib', 'reason': set([('filename', 'name')])},
15552                {'call': 'renameat2', 'reason': set([('filename', 'name')])},
15553                {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
15554                {'call': 'quotactl',
15555                 'reason': set([('filename', 'name'),
15556                                ('super_block', 's_flags'),
15557                                ('super_block', 's_qcop'),
15558                                ('super_block', 's_quota_types')])},
15559                {'call': 'acct', 'reason': set([('filename', 'name')])},
15560                {'call': 'open', 'reason': set([('filename', 'name')])},
15561                {'call': 'unlink', 'reason': set([('filename', 'name')])},
15562                {'call': 'rmdir', 'reason': set([('filename', 'name')])},
15563                {'call': 'swapon',
15564                 'reason': set([('filename', 'name'),
15565                                ('super_block', 's_flags'),
15566                                ('super_block', 's_qcop'),
15567                                ('super_block', 's_quota_types')])},
15568                {'call': 'mq_open', 'reason': set([('filename', 'name')])},
15569                {'call': 'unlinkat', 'reason': set([('filename', 'name')])}],
```

```
15570  'read': [{'call': 'syncfs',
15571            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15572           {'call': 'vmsplice',
15573            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15574           {'call': 'pwritev64',
15575            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15576           {'call': 'fremovexattr',
15577            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15578           {'call': 'readahead',
15579            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15580           {'call': 'getdents',
15581            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15582           {'call': 'writev',
15583            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15584           {'call': 'preadv64',
15585            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15586           {'call': 'fchmod',
15587            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15588           {'call': 'pread64',
15589            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15590           {'call': 'signalfd4',
15591            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15592           {'call': 'read', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15593           {'call': 'fchown',
15594            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15595           {'call': 'mq_timedreceive',
15596            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15597           {'call': 'utime', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15598           {'call': 'fsync', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15599           {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15600           {'call': 'recvfrom',
15601            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15602           {'call': 'fsetxattr',
15603            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15604           {'call': 'sendto',
15605            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15606           {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15607           {'call': 'sync_file_range',
15608            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15609           {'call': 'lseek', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15610           {'call': 'connect',
15611            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15612           {'call': 'getsockname',
15613            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15614           {'call': 'epoll_ctl',
15615            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15616           {'call': 'flock', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15617           {'call': 'pwritev',
15618            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15619           {'call': 'fchdir',
15620            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15621           {'call': 'accept4',
15622            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15623           {'call': 'old_readdir',
15624            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15625           {'call': 'inotify_rm_watch',
15626            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15627           {'call': 'utimensat',
15628            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15629           {'call': 'inotify_add_watch',
15630            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15631           {'call': 'preadv2',
15632            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15633           {'call': 'splice',
15634            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15635           {'call': 'ftruncate',
```



```
15636            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15637           {'call': 'preadv',
15638            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15639           {'call': 'getpeername',
15640            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15641           {'call': 'setsockopt',
15642            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15643           {'call': 'fcntl', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15644           {'call': 'ioctl', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15645           {'call': 'pwrite64',
15646            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15647           {'call': 'perf_event_open',
15648            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15649           {'call': 'pwritev64v2',
15650            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15651           {'call': 'futimesat',
15652            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15653           {'call': 'pwritev2',
15654            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15655           {'call': 'shutdown',
15656            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15657           {'call': 'getsockopt',
15658            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15659           {'call': 'mq_getsetattr',
15660            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15661           {'call': 'fdatasync',
15662            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15663           {'call': 'getdents64',
15664            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15665           {'call': 'listen',
15666            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15667           {'call': 'copy_file_range',
15668            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15669           {'call': 'mq_timedsend',
15670            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15671           {'call': 'fgetxattr',
15672            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15673           {'call': 'fcntl64',
15674            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15675           {'call': 'fallocate',
15676            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15677           {'call': 'epoll_wait',
15678            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15679           {'call': 'llseek',
15680            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15681           {'call': 'preadv64v2',
15682            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15683           {'call': 'readv', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15684           {'call': 'fstatfs',
15685            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15686           {'call': 'fstatfs64',
15687            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15688           {'call': 'write', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15689           {'call': 'mq_notify',
15690            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15691           {'call': 'sendfile',
15692            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15693           {'call': 'bind', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15694           {'call': 'flistxattr',
15695            'reason': set([('fd', 'file'), ('fd', 'flags')])},
15696           {'call': 'sendfile64',
15697            'reason': set([('fd', 'file'), ('fd', 'flags')])}],
15698  'readahead': [{'call': 'syncfs',
15699                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15700                {'call': 'vmsplice',
15701                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```

```
15702                {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
15703                {'call': 'mq_unlink',
15704                 'reason': set([('address_space', 'a_ops')])},
15705                {'call': 'pwritev64',
15706                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15707                {'call': 'swapoff',
15708                 'reason': set([('address_space', 'a_ops'),
15709                                ('file', 'f_mode')])},
15710                {'call': 'fremovexattr',
15711                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15712                {'call': 'readahead',
15713                 'reason': set([('address_space', 'a_ops'),
15714                                ('fd', 'file'),
15715                                ('fd', 'flags')])},
15716                {'call': 'getdents',
15717                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15718                {'call': 'writev',
15719                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15720                {'call': 'preadv64',
15721                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15722                {'call': 'fchmod',
15723                 'reason': set([('address_space', 'a_ops'),
15724                                ('fd', 'file'),
15725                                ('fd', 'flags')])},
15726                {'call': 'pread64',
15727                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15728                {'call': 'signalfd4',
15729                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15730                {'call': 'memfd_create',
15731                 'reason': set([('address_space', 'a_ops'),
15732                                ('file', 'f_mode')])},
15733                {'call': 'remap_file_pages',
15734                 'reason': set([('file', 'f_mode')])},
15735                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
15736                {'call': 'readlinkat',
15737                 'reason': set([('address_space', 'a_ops')])},
15738                {'call': 'read',
15739                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15740                {'call': 'fchown',
15741                 'reason': set([('address_space', 'a_ops'),
15742                                ('fd', 'file'),
15743                                ('fd', 'flags')])},
15744                {'call': 'mq_timedreceive',
15745                 'reason': set([('address_space', 'a_ops'),
15746                                ('fd', 'file'),
15747                                ('fd', 'flags')])},
15748                {'call': 'utime',
15749                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15750                {'call': 'fsync',
15751                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15752                {'call': 'bpf',
15753                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15754                {'call': 'recvfrom',
15755                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15756                {'call': 'fsetxattr',
15757                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15758                {'call': 'sendto',
15759                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15760                {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
15761                {'call': 'tee',
15762                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15763                {'call': 'sync_file_range',
15764                 'reason': set([('address_space', 'a_ops'),
15765                                ('fd', 'file'),
15766                                ('fd', 'flags')])},
15767                {'call': 'lseek',
```



```
15768                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15769                {'call': 'connect',
15770                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15771                {'call': 'getsockname',
15772                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15773                {'call': 'epoll_ctl',
15774                 'reason': set([('fd', 'file'),
15775                                ('fd', 'flags'),
15776                                ('file', 'f_mode')])},
15777                {'call': 'flock',
15778                 'reason': set([('fd', 'file'),
15779                                ('fd', 'flags'),
15780                                ('file', 'f_mode')])},
15781                {'call': 'pwritev',
15782                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15783                {'call': 'fchdir',
15784                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15785                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
15786                {'call': 'uselib',
15787                 'reason': set([('address_space', 'a_ops'),
15788                                ('file', 'f_mode')])},
15789                {'call': 'accept4',
15790                 'reason': set([('fd', 'file'),
15791                                ('fd', 'flags'),
15792                                ('file', 'f_mode')])},
15793                {'call': 'old_readdir',
15794                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15795                {'call': 'inotify_rm_watch',
15796                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15797                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
15798                {'call': 'utimensat',
15799                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15800                {'call': 'fchmodat',
15801                 'reason': set([('address_space', 'a_ops')])},
15802                {'call': 'inotify_add_watch',
15803                 'reason': set([('address_space', 'a_ops'),
15804                                ('fd', 'file'),
15805                                ('fd', 'flags')])},
15806                {'call': 'preadv2',
15807                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15808                {'call': 'splice',
15809                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15810                {'call': 'ftruncate',
15811                 'reason': set([('address_space', 'a_ops'),
15812                                ('fd', 'file'),
15813                                ('fd', 'flags')])},
15814                {'call': 'preadv',
15815                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15816                {'call': 'getpeername',
15817                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15818                {'call': 'shmat',
15819                 'reason': set([('address_space', 'a_ops'),
15820                                ('file', 'f_mode')])},
15821                {'call': 'setsockopt',
15822                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15823                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
15824                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
15825                {'call': 'fcntl',
15826                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15827                {'call': 'ioctl',
15828                 'reason': set([('address_space', 'a_ops'),
15829                                ('fd', 'file'),
15830                                ('fd', 'flags')])},
15831                {'call': 'pwrite64',
15832                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
15833                {'call': 'perf_event_open',
```

```
                 'reason': set([('fd', 'file'),
                                ('fd', 'flags'),
                                ('file', 'f_mode')])},
               {'call': 'linkat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
               {'call': 'pwritev64v2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'futimesat',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'pwritev2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'shutdown',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
               {'call': 'open', 'reason': set([('file', 'f_mode')])},
               {'call': 'unlink',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'getsockopt',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'mq_getsetattr',
                'reason': set([('address_space', 'a_ops'),
                               ('fd', 'file'),
                               ('fd', 'flags')])},
               {'call': 'faccessat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
               {'call': 'fdatasync',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
               {'call': 'getdents64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'listen',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'copy_file_range',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'mq_timedsend',
                'reason': set([('address_space', 'a_ops'),
                               ('fd', 'file'),
                               ('fd', 'flags')])},
               {'call': 'fgetxattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
               {'call': 'fcntl64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'swapon',
                'reason': set([('address_space', 'a_ops'),
                               ('file', 'f_mode')])},
               {'call': 'fallocate',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'epoll_wait',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fchownat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'llseek',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
               {'call': 'preadv64v2',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'readv',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fstatfs',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'fstatfs64',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'write',
```



```
                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'mq_notify',
                'reason': set([('address_space', 'a_ops'),
                               ('fd', 'file'),
                               ('fd', 'flags')])},
               {'call': 'sendfile',
                'reason': set([('address_space', 'a_ops'),
                               ('fd', 'file'),
                               ('fd', 'flags')])},
               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
               {'call': 'unlinkat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'open_by_handle_at',
                'reason': set([('file', 'f_mode')])},
               {'call': 'bind',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'flistxattr',
                'reason': set([('fd', 'file'), ('fd', 'flags')])},
               {'call': 'sendfile64',
                'reason': set([('address_space', 'a_ops'),
                               ('fd', 'file'),
                               ('fd', 'flags')])}],
 'readlinkat': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
                {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
                {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
                {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
                {'call': 'remap_file_pages',
                 'reason': set([('path', 'dentry')])},
                {'call': 'dup3', 'reason': set([('path', 'dentry')])},
                {'call': 'unshare', 'reason': set([('path', 'dentry')])},
                {'call': 'epoll_create1',
                 'reason': set([('path', 'dentry')])},
                {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
                {'call': 'flock', 'reason': set([('path', 'dentry')])},
                {'call': 'openat', 'reason': set([('path', 'dentry')])},
                {'call': 'lookup_dcookie',
                 'reason': set([('path', 'dentry')])},
                {'call': 'uselib', 'reason': set([('path', 'dentry')])},
                {'call': 'accept4', 'reason': set([('path', 'dentry')])},
                {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
                {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
                {'call': 'shmat', 'reason': set([('path', 'dentry')])},
                {'call': 'socket', 'reason': set([('path', 'dentry')])},
                {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
                {'call': 'perf_event_open',
                 'reason': set([('path', 'dentry')])},
                {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
                {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
                {'call': 'acct', 'reason': set([('path', 'dentry')])},
                {'call': 'open', 'reason': set([('path', 'dentry')])},
                {'call': 'dup', 'reason': set([('path', 'dentry')])},
                {'call': 'setns', 'reason': set([('path', 'dentry')])},
                {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
                {'call': 'swapon', 'reason': set([('path', 'dentry')])},
                {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
                {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('path', 'dentry')])}],
 'readv': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
           {'call': 'readahead', 'reason': set([('fd', 'file')])},
           {'call': 'getdents', 'reason': set([('fd', 'file')])},
           {'call': 'writev', 'reason': set([('fd', 'file')])},
           {'call': 'preadv64', 'reason': set([('fd', 'file')])},
```

```
15966             {'call': 'fchmod', 'reason': set([('fd', 'file')])},
15967             {'call': 'pread64', 'reason': set([('fd', 'file')])},
15968             {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
15969             {'call': 'read', 'reason': set([('fd', 'file')])},
15970             {'call': 'fchown', 'reason': set([('fd', 'file')])},
15971             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
15972             {'call': 'utime', 'reason': set([('fd', 'file')])},
15973             {'call': 'fsync', 'reason': set([('fd', 'file')])},
15974             {'call': 'bpf', 'reason': set([('fd', 'file')])},
15975             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
15976             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
15977             {'call': 'sendto', 'reason': set([('fd', 'file')])},
15978             {'call': 'tee', 'reason': set([('fd', 'file')])},
15979             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
15980             {'call': 'lseek', 'reason': set([('fd', 'file')])},
15981             {'call': 'connect', 'reason': set([('fd', 'file')])},
15982             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
15983             {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
15984             {'call': 'flock', 'reason': set([('fd', 'file')])},
15985             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
15986             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
15987             {'call': 'accept4', 'reason': set([('fd', 'file')])},
15988             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
15989             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
15990             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
15991             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
15992             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
15993             {'call': 'splice', 'reason': set([('fd', 'file')])},
15994             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
15995             {'call': 'preadv', 'reason': set([('fd', 'file')])},
15996             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
15997             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
15998             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
15999             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
16000             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
16001             {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
16002             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
16003             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
16004             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
16005             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
16006             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
16007             {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
16008             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
16009             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
16010             {'call': 'listen', 'reason': set([('fd', 'file')])},
16011             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
16012             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
16013             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
16014             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
16015             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
16016             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
16017             {'call': 'llseek', 'reason': set([('fd', 'file')])},
16018             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
16019             {'call': 'readv', 'reason': set([('fd', 'file')])},
16020             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
16021             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
16022             {'call': 'write', 'reason': set([('fd', 'file')])},
16023             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
16024             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
16025             {'call': 'bind', 'reason': set([('fd', 'file')])},
16026             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
16027             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
16028  'reboot': [{'call': 'reboot', 'reason': set([('pid_namespace', 'user_ns')])},
16029             {'call': 'perf_event_open',
16030              'reason': set([('pid_namespace', 'user_ns')])},
16031             {'call': 'acct', 'reason': set([('pid_namespace', 'user_ns')])},
```



```
16032             {'call': 'setns', 'reason': set([('pid_namespace', 'user_ns')])}],
16033  'recvfrom': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
16034               {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
16035               {'call': 'eventfd2', 'reason': set([('file', 'f_flags')])},
16036               {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
16037               {'call': 'swapoff', 'reason': set([('file', 'f_flags')])},
16038               {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
16039               {'call': 'readahead', 'reason': set([('fd', 'file')])},
16040               {'call': 'getdents', 'reason': set([('fd', 'file')])},
16041               {'call': 'writev', 'reason': set([('fd', 'file')])},
16042               {'call': 'preadv64', 'reason': set([('fd', 'file')])},
16043               {'call': 'fchmod', 'reason': set([('fd', 'file')])},
16044               {'call': 'pread64', 'reason': set([('fd', 'file')])},
16045               {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
16046               {'call': 'memfd_create', 'reason': set([('file', 'f_flags')])},
16047               {'call': 'remap_file_pages',
16048                'reason': set([('file', 'f_flags')])},
16049               {'call': 'dup3', 'reason': set([('file', 'f_flags')])},
16050               {'call': 'read', 'reason': set([('fd', 'file')])},
16051               {'call': 'fchown', 'reason': set([('fd', 'file')])},
16052               {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
16053               {'call': 'utime', 'reason': set([('fd', 'file')])},
16054               {'call': 'fsync', 'reason': set([('fd', 'file')])},
16055               {'call': 'bpf', 'reason': set([('fd', 'file')])},
16056               {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
16057               {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
16058               {'call': 'sendto', 'reason': set([('fd', 'file')])},
16059               {'call': 'epoll_create1', 'reason': set([('file', 'f_flags')])},
16060               {'call': 'tee', 'reason': set([('fd', 'file')])},
16061               {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
16062               {'call': 'lseek', 'reason': set([('fd', 'file')])},
16063               {'call': 'connect', 'reason': set([('fd', 'file')])},
16064               {'call': 'getsockname', 'reason': set([('fd', 'file')])},
16065               {'call': 'epoll_ctl',
16066                'reason': set([('fd', 'file'), ('file', 'f_flags')])},
16067               {'call': 'flock',
16068                'reason': set([('fd', 'file'), ('file', 'f_flags')])},
16069               {'call': 'pwritev', 'reason': set([('fd', 'file')])},
16070               {'call': 'fchdir', 'reason': set([('fd', 'file')])},
16071               {'call': 'openat', 'reason': set([('file', 'f_flags')])},
16072               {'call': 'uselib', 'reason': set([('file', 'f_flags')])},
16073               {'call': 'accept4',
16074                'reason': set([('fd', 'file'), ('file', 'f_flags')])},
16075               {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
16076               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
16077               {'call': 'socketpair', 'reason': set([('file', 'f_flags')])},
16078               {'call': 'utimensat', 'reason': set([('fd', 'file')])},
16079               {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
16080               {'call': 'preadv2', 'reason': set([('fd', 'file')])},
16081               {'call': 'splice', 'reason': set([('fd', 'file')])},
16082               {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
16083               {'call': 'preadv', 'reason': set([('fd', 'file')])},
16084               {'call': 'getpeername', 'reason': set([('fd', 'file')])},
16085               {'call': 'shmat', 'reason': set([('file', 'f_flags')])},
16086               {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
16087               {'call': 'socket', 'reason': set([('file', 'f_flags')])},
16088               {'call': 'pipe2', 'reason': set([('file', 'f_flags')])},
16089               {'call': 'fcntl', 'reason': set([('fd', 'file')])},
16090               {'call': 'ioctl', 'reason': set([('fd', 'file')])},
16091               {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
16092               {'call': 'perf_event_open',
16093                'reason': set([('fd', 'file'), ('file', 'f_flags')])},
16094               {'call': 'shmdt', 'reason': set([('file', 'f_flags')])},
16095               {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
16096               {'call': 'futimesat', 'reason': set([('fd', 'file')])},
16097               {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
```

```
16098               {'call': 'shutdown', 'reason': set([('fd', 'file')])},
16099               {'call': 'acct', 'reason': set([('file', 'f_flags')])},
16100               {'call': 'open', 'reason': set([('file', 'f_flags')])},
16101               {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
16102               {'call': 'mq_getsetattr',
16103                'reason': set([('fd', 'file'), ('file', 'f_flags')])},
16104               {'call': 'dup', 'reason': set([('file', 'f_flags')])},
16105               {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
16106               {'call': 'setns', 'reason': set([('file', 'f_flags')])},
16107               {'call': 'getdents64', 'reason': set([('fd', 'file')])},
16108               {'call': 'listen', 'reason': set([('fd', 'file')])},
16109               {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
16110               {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
16111               {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
16112               {'call': 'shmctl', 'reason': set([('file', 'f_flags')])},
16113               {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
16114               {'call': 'swapon', 'reason': set([('file', 'f_flags')])},
16115               {'call': 'fallocate', 'reason': set([('fd', 'file')])},
16116               {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
16117               {'call': 'llseek', 'reason': set([('fd', 'file')])},
16118               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_flags')])},
16119               {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
16120               {'call': 'readv', 'reason': set([('fd', 'file')])},
16121               {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
16122               {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
16123               {'call': 'write', 'reason': set([('fd', 'file')])},
16124               {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
16125               {'call': 'sendfile', 'reason': set([('fd', 'file')])},
16126               {'call': 'mq_open', 'reason': set([('file', 'f_flags')])},
16127               {'call': 'open_by_handle_at',
16128                'reason': set([('file', 'f_flags')])},
16129               {'call': 'bind', 'reason': set([('fd', 'file')])},
16130               {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
16131               {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
16132  'recvmmsg': [{'call': 'rt_sigtimedwait',
16133                'reason': set([('timespec', 'tv_nsec'),
16134                               ('timespec', 'tv_sec')])},
16135               {'call': 'mq_unlink',
16136                'reason': set([('timespec', 'tv_nsec'),
16137                               ('timespec', 'tv_sec')])},
16138               {'call': 'swapoff',
16139                'reason': set([('timespec', 'tv_nsec'),
16140                               ('timespec', 'tv_sec')])},
16141               {'call': 'fchmod',
16142                'reason': set([('timespec', 'tv_nsec'),
16143                               ('timespec', 'tv_sec')])},
16144               {'call': 'memfd_create',
16145                'reason': set([('timespec', 'tv_nsec'),
16146                               ('timespec', 'tv_sec')])},
16147               {'call': 'readlinkat',
16148                'reason': set([('timespec', 'tv_nsec'),
16149                               ('timespec', 'tv_sec')])},
16150               {'call': 'io_getevents',
16151                'reason': set([('timespec', 'tv_nsec'),
16152                               ('timespec', 'tv_sec')])},
16153               {'call': 'fchown',
16154                'reason': set([('timespec', 'tv_nsec'),
16155                               ('timespec', 'tv_sec')])},
16156               {'call': 'mq_timedreceive',
16157                'reason': set([('timespec', 'tv_nsec'),
16158                               ('timespec', 'tv_sec')])},
16159               {'call': 'utime',
16160                'reason': set([('timespec', 'tv_nsec'),
16161                               ('timespec', 'tv_sec')])},
16162               {'call': 'semtimedop',
16163                'reason': set([('timespec', 'tv_nsec'),
```



```
16164                               ('timespec', 'tv_sec')])},
16165               {'call': 'recvfrom',
16166                'reason': set([('msghdr', 'msg_flags'),
16167                               ('socket', 'file'),
16168                               ('socket', 'sk')])},
16169               {'call': 'settimeofday',
16170                'reason': set([('timespec', 'tv_nsec'),
16171                               ('timespec', 'tv_sec')])},
16172               {'call': 'sendto',
16173                'reason': set([('msghdr', 'msg_flags'),
16174                               ('socket', 'file'),
16175                               ('socket', 'sk')])},
16176               {'call': 'sched_rr_get_interval',
16177                'reason': set([('timespec', 'tv_nsec'),
16178                               ('timespec', 'tv_sec')])},
16179               {'call': 'timerfd_gettime',
16180                'reason': set([('timespec', 'tv_nsec'),
16181                               ('timespec', 'tv_sec')])},
16182               {'call': 'connect',
16183                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16184               {'call': 'getsockname',
16185                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16186               {'call': 'pselect6',
16187                'reason': set([('timespec', 'tv_nsec'),
16188                               ('timespec', 'tv_sec')])},
16189               {'call': 'uselib',
16190                'reason': set([('timespec', 'tv_nsec'),
16191                               ('timespec', 'tv_sec')])},
16192               {'call': 'accept4',
16193                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16194               {'call': 'fchmodat',
16195                'reason': set([('timespec', 'tv_nsec'),
16196                               ('timespec', 'tv_sec')])},
16197               {'call': 'inotify_add_watch',
16198                'reason': set([('timespec', 'tv_nsec'),
16199                               ('timespec', 'tv_sec')])},
16200               {'call': 'timer_settime',
16201                'reason': set([('timespec', 'tv_nsec'),
16202                               ('timespec', 'tv_sec')])},
16203               {'call': 'ftruncate',
16204                'reason': set([('timespec', 'tv_nsec'),
16205                               ('timespec', 'tv_sec')])},
16206               {'call': 'timer_gettime',
16207                'reason': set([('timespec', 'tv_nsec'),
16208                               ('timespec', 'tv_sec')])},
16209               {'call': 'getpeername',
16210                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16211               {'call': 'setsockopt',
16212                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16213               {'call': 'ioctl',
16214                'reason': set([('timespec', 'tv_nsec'),
16215                               ('timespec', 'tv_sec')])},
16216               {'call': 'linkat',
16217                'reason': set([('timespec', 'tv_nsec'),
16218                               ('timespec', 'tv_sec')])},
16219               {'call': 'sendmsg',
16220                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16221               {'call': 'stime',
16222                'reason': set([('timespec', 'tv_nsec'),
16223                               ('timespec', 'tv_sec')])},
16224               {'call': 'futimesat',
16225                'reason': set([('timespec', 'tv_nsec'),
16226                               ('timespec', 'tv_sec')])},
16227               {'call': 'shutdown',
16228                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16229               {'call': 'poll',
```

```
16230                 'reason': set([('timespec', 'tv_nsec'),
16231                                ('timespec', 'tv_sec')])},
16232               {'call': 'select',
16233                'reason': set([('timespec', 'tv_nsec'),
16234                               ('timespec', 'tv_sec')])},
16235               {'call': 'unlink',
16236                'reason': set([('timespec', 'tv_nsec'),
16237                               ('timespec', 'tv_sec')])},
16238               {'call': 'getsockopt',
16239                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16240               {'call': 'nanosleep',
16241                'reason': set([('timespec', 'tv_nsec'),
16242                               ('timespec', 'tv_sec')])},
16243               {'call': 'mq_getsetattr',
16244                'reason': set([('timespec', 'tv_nsec'),
16245                               ('timespec', 'tv_sec')])},
16246               {'call': 'faccessat',
16247                'reason': set([('timespec', 'tv_nsec'),
16248                               ('timespec', 'tv_sec')])},
16249               {'call': 'listen',
16250                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16251               {'call': 'mq_timedsend',
16252                'reason': set([('timespec', 'tv_nsec'),
16253                               ('timespec', 'tv_sec')])},
16254               {'call': 'swapon',
16255                'reason': set([('timespec', 'tv_nsec'),
16256                               ('timespec', 'tv_sec')])},
16257               {'call': 'epoll_wait',
16258                'reason': set([('timespec', 'tv_nsec'),
16259                               ('timespec', 'tv_sec')])},
16260               {'call': 'fchownat',
16261                'reason': set([('timespec', 'tv_nsec'),
16262                               ('timespec', 'tv_sec')])},
16263               {'call': 'fstat',
16264                'reason': set([('timespec', 'tv_nsec'),
16265                               ('timespec', 'tv_sec')])},
16266               {'call': 'timerfd_settime',
16267                'reason': set([('timespec', 'tv_nsec'),
16268                               ('timespec', 'tv_sec')])},
16269               {'call': 'recvmsg',
16270                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16271               {'call': 'mq_notify',
16272                'reason': set([('timespec', 'tv_nsec'),
16273                               ('timespec', 'tv_sec')])},
16274               {'call': 'sendfile',
16275                'reason': set([('timespec', 'tv_nsec'),
16276                               ('timespec', 'tv_sec')])},
16277               {'call': 'sendmmsg',
16278                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16279               {'call': 'newfstat',
16280                'reason': set([('timespec', 'tv_nsec'),
16281                               ('timespec', 'tv_sec')])},
16282               {'call': 'clock_nanosleep',
16283                'reason': set([('timespec', 'tv_nsec'),
16284                               ('timespec', 'tv_sec')])},
16285               {'call': 'unlinkat',
16286                'reason': set([('timespec', 'tv_nsec'),
16287                               ('timespec', 'tv_sec')])},
16288               {'call': 'bind',
16289                'reason': set([('socket', 'file'), ('socket', 'sk')])},
16290               {'call': 'futex',
16291                'reason': set([('timespec', 'tv_nsec'),
16292                               ('timespec', 'tv_sec')])},
16293               {'call': 'recvmmsg',
16294                'reason': set([('socket', 'file'),
16295                               ('socket', 'sk'),
```



```
16296                               ('timespec', 'tv_nsec'),
16297                               ('timespec', 'tv_sec')])},
16298             {'call': 'sendfile64',
16299              'reason': set([('timespec', 'tv_nsec'),
16300                             ('timespec', 'tv_sec')])},
16301             {'call': 'ppoll',
16302              'reason': set([('timespec', 'tv_nsec'),
16303                             ('timespec', 'tv_sec')])}],
16304  'recvmsg': [{'call': 'recvfrom', 'reason': set([('socket', 'file')])},
16305              {'call': 'sendto', 'reason': set([('socket', 'file')])},
16306              {'call': 'connect', 'reason': set([('socket', 'file')])},
16307              {'call': 'getsockname', 'reason': set([('socket', 'file')])},
16308              {'call': 'accept4', 'reason': set([('socket', 'file')])},
16309              {'call': 'getpeername', 'reason': set([('socket', 'file')])},
16310              {'call': 'setsockopt', 'reason': set([('socket', 'file')])},
16311              {'call': 'sendmsg', 'reason': set([('socket', 'file')])},
16312              {'call': 'shutdown', 'reason': set([('socket', 'file')])},
16313              {'call': 'getsockopt', 'reason': set([('socket', 'file')])},
16314              {'call': 'listen', 'reason': set([('socket', 'file')])},
16315              {'call': 'recvmsg', 'reason': set([('socket', 'file')])},
16316              {'call': 'sendmmsg', 'reason': set([('socket', 'file')])},
16317              {'call': 'bind', 'reason': set([('socket', 'file')])},
16318              {'call': 'recvmmsg', 'reason': set([('socket', 'file')])}],
16319  'remap_file_pages': [{'call': 'remap_file_pages',
16320                        'reason': set([('vm_area_struct', 'vm_end'),
16321                                       ('vm_area_struct', 'vm_file'),
16322                                       ('vm_area_struct', 'vm_flags'),
16323                                       ('vm_area_struct', 'vm_start')])},
16324                       {'call': 'shmdt',
16325                        'reason': set([('vm_area_struct', 'vm_end'),
16326                                       ('vm_area_struct', 'vm_file'),
16327                                       ('vm_area_struct', 'vm_flags'),
16328                                       ('vm_area_struct', 'vm_start')])},
16329                       {'call': 'brk',
16330                        'reason': set([('vm_area_struct', 'vm_end'),
16331                                       ('vm_area_struct', 'vm_file'),
16332                                       ('vm_area_struct', 'vm_flags'),
16333                                       ('vm_area_struct', 'vm_start')])},
16334                       {'call': 'get_mempolicy',
16335                        'reason': set([('vm_area_struct', 'vm_end'),
16336                                       ('vm_area_struct', 'vm_file'),
16337                                       ('vm_area_struct', 'vm_flags'),
16338                                       ('vm_area_struct', 'vm_start')])},
16339                       {'call': 'munlockall',
16340                        'reason': set([('vm_area_struct', 'vm_end'),
16341                                       ('vm_area_struct', 'vm_file'),
16342                                       ('vm_area_struct', 'vm_flags'),
16343                                       ('vm_area_struct', 'vm_start')])},
16344                       {'call': 'pkey_mprotect',
16345                        'reason': set([('vm_area_struct', 'vm_end'),
16346                                       ('vm_area_struct', 'vm_file'),
16347                                       ('vm_area_struct', 'vm_flags'),
16348                                       ('vm_area_struct', 'vm_start')])},
16349                       {'call': 'madvise',
16350                        'reason': set([('vm_area_struct', 'vm_end'),
16351                                       ('vm_area_struct', 'vm_file'),
16352                                       ('vm_area_struct', 'vm_flags'),
16353                                       ('vm_area_struct', 'vm_start')])},
16354                       {'call': 'mprotect',
16355                        'reason': set([('vm_area_struct', 'vm_end'),
16356                                       ('vm_area_struct', 'vm_file'),
16357                                       ('vm_area_struct', 'vm_flags'),
16358                                       ('vm_area_struct', 'vm_start')])},
16359                       {'call': 'mremap',
16360                        'reason': set([('vm_area_struct', 'vm_end'),
16361                                       ('vm_area_struct', 'vm_file'),
```

```
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])},
                    {'call': 'prctl',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_file'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'munlock',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_file'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'mincore',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_file'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])},
                    {'call': 'mlockall',
                     'reason': set([('vm_area_struct', 'vm_end'),
                                    ('vm_area_struct', 'vm_file'),
                                    ('vm_area_struct', 'vm_flags'),
                                    ('vm_area_struct', 'vm_start')])}],
 'removexattr': [{'call': 'eventfd2',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'swapoff',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'pivot_root',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'memfd_create',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'remap_file_pages',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'dup3',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'unshare',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'epoll_create1',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'epoll_ctl',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'flock',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'openat',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'lookup_dcookie',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'uselib',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'accept4',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'socketpair',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'getcwd',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'shmat',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'socket',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'pipe2',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'perf_event_open',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'shmdt',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'quotactl',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```



```
                 {'call': 'acct',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'open',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'dup',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'setns',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'shmctl',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'swapon',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'mmap_pgoff',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'mq_open',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'open_by_handle_at',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'renameat2': [{'call': 'sysfs',
                'reason': set([('filename', 'name'),
                               ('filename', 'refcnt')])},
               {'call': 'eventfd2',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mq_unlink',
                'reason': set([('dentry', 'd_inode'),
                               ('filename', 'name'),
                               ('filename', 'refcnt')])},
               {'call': 'swapoff',
                'reason': set([('filename', 'name'),
                               ('filename', 'refcnt'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'pivot_root',
                'reason': set([('dentry', 'd_inode'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'memfd_create',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'remap_file_pages',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'dup3',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'unshare',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'epoll_create1',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'epoll_ctl',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'flock',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'openat',
                'reason': set([('filename', 'name'),
                               ('filename', 'refcnt'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'lookup_dcookie',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'uselib',
                'reason': set([('filename', 'name'),
                               ('filename', 'refcnt'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'renameat2',
                'reason': set([('dentry', 'd_inode'),
                               ('filename', 'name'),
```

```
                                ('filename', 'refcnt')])},
                {'call': 'accept4',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'socketpair',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'getcwd',
                 'reason': set([('dentry', 'd_inode'),
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
                {'call': 'shmat',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
                {'call': 'socket',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'symlinkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'pipe2',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'perf_event_open',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
                {'call': 'shmdt',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'quotactl',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt'),
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'acct',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt'),
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'open',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt'),
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'unlink',
                 'reason': set([('dentry', 'd_inode'),
                                ('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'rmdir',
                 'reason': set([('dentry', 'd_inode'),
                                ('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'dup',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'setns',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'shmctl',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'swapon',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt'),
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'mmap_pgoff',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mq_open',
                 'reason': set([('dentry', 'd_inode'),
                                ('filename', 'name'),
                                ('filename', 'refcnt'),
```



```
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'unlinkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'rmdir': [{'call': 'eventfd2',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'mq_unlink', 'reason': set([('dentry', 'd_inode')])},
           {'call': 'swapoff',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'pivot_root',
            'reason': set([('dentry', 'd_inode'),
                           ('path', 'dentry'),
                           ('path', 'mnt')])},
           {'call': 'memfd_create',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'remap_file_pages',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'dup3',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'unshare',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
           {'call': 'epoll_create1',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'epoll_ctl',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'flock',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'openat',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'lookup_dcookie',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'uselib',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'renameat2', 'reason': set([('dentry', 'd_inode')])},
           {'call': 'accept4',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'socketpair',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'getcwd',
            'reason': set([('dentry', 'd_inode'),
                           ('path', 'dentry'),
                           ('path', 'mnt')])},
           {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
           {'call': 'shmat',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
           {'call': 'socket',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'symlinkat', 'reason': set([('dentry', 'd_inode')])},
           {'call': 'pipe2',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'perf_event_open',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
           {'call': 'shmdt',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'quotactl',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'acct',
            'reason': set([('path', 'dentry'), ('path', 'mnt')])},
           {'call': 'open',
```

```
16626             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
16627            {'call': 'unlink', 'reason': set([('dentry', 'd_inode')])},
16628            {'call': 'rmdir', 'reason': set([('dentry', 'd_inode')])},
16629            {'call': 'dup',
16630             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
16631            {'call': 'setns',
16632             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
16633            {'call': 'shmctl',
16634             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
16635            {'call': 'swapon',
16636             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
16637            {'call': 'mmap_pgoff',
16638             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
16639            {'call': 'mq_open',
16640             'reason': set([('dentry', 'd_inode'),
16641                            ('path', 'dentry'),
16642                            ('path', 'mnt')])},
16643            {'call': 'unlinkat', 'reason': set([('dentry', 'd_inode')])},
16644            {'call': 'open_by_handle_at',
16645             'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
16646  'rt_sigqueueinfo': [{'call': 'rt_sigtimedwait',
16647                       'reason': set([('siginfo', 'si_code')])},
16648                      {'call': 'kill',
16649                       'reason': set([('siginfo', 'si_code')])},
16650                      {'call': 'timer_create',
16651                       'reason': set([('siginfo', 'si_code')])},
16652                      {'call': 'rt_sigqueueinfo',
16653                       'reason': set([('siginfo', 'si_code')])},
16654                      {'call': 'tgkill',
16655                       'reason': set([('siginfo', 'si_code')])},
16656                      {'call': 'rt_tgsigqueueinfo',
16657                       'reason': set([('siginfo', 'si_code')])},
16658                      {'call': 'rt_sigreturn',
16659                       'reason': set([('siginfo', 'si_code')])},
16660                      {'call': 'tkill',
16661                       'reason': set([('siginfo', 'si_code')])}],
16662  'rt_sigreturn': [{'call': 'keyctl',
16663                    'reason': set([('mm_segment_t', 'seg'),
16664                                   ('thread_struct', 'uaccess_err')])},
16665                   {'call': 'rt_sigtimedwait',
16666                    'reason': set([('mm_segment_t', 'seg'),
16667                                   ('thread_struct', 'uaccess_err')])},
16668                   {'call': 'msgrcv',
16669                    'reason': set([('mm_segment_t', 'seg'),
16670                                   ('thread_struct', 'uaccess_err')])},
16671                   {'call': 'kill',
16672                    'reason': set([('mm_segment_t', 'seg'),
16673                                   ('thread_struct', 'uaccess_err')])},
16674                   {'call': 'sched_getaffinity',
16675                    'reason': set([('mm_segment_t', 'seg'),
16676                                   ('thread_struct', 'uaccess_err')])},
16677                   {'call': 'sched_setparam',
16678                    'reason': set([('mm_segment_t', 'seg'),
16679                                   ('thread_struct', 'uaccess_err')])},
16680                   {'call': 'ioprio_set',
16681                    'reason': set([('mm_segment_t', 'seg'),
16682                                   ('thread_struct', 'uaccess_err')])},
16683                   {'call': 'getppid',
16684                    'reason': set([('mm_segment_t', 'seg'),
16685                                   ('thread_struct', 'uaccess_err')])},
16686                   {'call': 'ioperm',
16687                    'reason': set([('mm_segment_t', 'seg'),
16688                                   ('thread_struct', 'uaccess_err')])},
16689                   {'call': 'mq_timedreceive',
16690                    'reason': set([('mm_segment_t', 'seg'),
16691                                   ('thread_struct', 'uaccess_err')])},
```



```
16692                   {'call': 'capget',
16693                    'reason': set([('mm_segment_t', 'seg'),
16694                                   ('thread_struct', 'uaccess_err')])},
16695                   {'call': 'sched_setaffinity',
16696                    'reason': set([('mm_segment_t', 'seg'),
16697                                   ('thread_struct', 'uaccess_err')])},
16698                   {'call': 'signal',
16699                    'reason': set([('mm_segment_t', 'seg'),
16700                                   ('thread_struct', 'uaccess_err')])},
16701                   {'call': 'semtimedop',
16702                    'reason': set([('mm_segment_t', 'seg'),
16703                                   ('thread_struct', 'uaccess_err')])},
16704                   {'call': 'umount',
16705                    'reason': set([('mm_segment_t', 'seg'),
16706                                   ('thread_struct', 'uaccess_err')])},
16707                   {'call': 'sched_rr_get_interval',
16708                    'reason': set([('mm_segment_t', 'seg'),
16709                                   ('thread_struct', 'uaccess_err')])},
16710                   {'call': 'rt_sigprocmask',
16711                    'reason': set([('mm_segment_t', 'seg'),
16712                                   ('thread_struct', 'uaccess_err')])},
16713                   {'call': 'setsid',
16714                    'reason': set([('mm_segment_t', 'seg'),
16715                                   ('thread_struct', 'uaccess_err')])},
16716                   {'call': 'sigaltstack',
16717                    'reason': set([('mm_segment_t', 'seg'),
16718                                   ('thread_struct', 'uaccess_err')])},
16719                   {'call': 'sched_setattr',
16720                    'reason': set([('mm_segment_t', 'seg'),
16721                                   ('thread_struct', 'uaccess_err')])},
16722                   {'call': 'migrate_pages',
16723                    'reason': set([('mm_segment_t', 'seg'),
16724                                   ('thread_struct', 'uaccess_err')])},
16725                   {'call': 'getitimer',
16726                    'reason': set([('mm_segment_t', 'seg'),
16727                                   ('thread_struct', 'uaccess_err')])},
16728                   {'call': 'setpgid',
16729                    'reason': set([('mm_segment_t', 'seg'),
16730                                   ('thread_struct', 'uaccess_err')])},
16731                   {'call': 'getsid',
16732                    'reason': set([('mm_segment_t', 'seg'),
16733                                   ('thread_struct', 'uaccess_err')])},
16734                   {'call': 'prlimit64',
16735                    'reason': set([('mm_segment_t', 'seg'),
16736                                   ('thread_struct', 'uaccess_err')])},
16737                   {'call': 'perf_event_open',
16738                    'reason': set([('mm_segment_t', 'seg'),
16739                                   ('thread_struct', 'uaccess_err')])},
16740                   {'call': 'rt_sigaction',
16741                    'reason': set([('mm_segment_t', 'seg'),
16742                                   ('thread_struct', 'uaccess_err')])},
16743                   {'call': 'getpgid',
16744                    'reason': set([('mm_segment_t', 'seg'),
16745                                   ('thread_struct', 'uaccess_err')])},
16746                   {'call': 'getpriority',
16747                    'reason': set([('mm_segment_t', 'seg'),
16748                                   ('thread_struct', 'uaccess_err')])},
16749                   {'call': 'sigaction',
16750                    'reason': set([('mm_segment_t', 'seg'),
16751                                   ('thread_struct', 'uaccess_err')])},
16752                   {'call': 'setns',
16753                    'reason': set([('mm_segment_t', 'seg'),
16754                                   ('thread_struct', 'uaccess_err')])},
16755                   {'call': 'fork',
16756                    'reason': set([('mm_segment_t', 'seg'),
16757                                   ('thread_struct', 'uaccess_err')])},
```

```
                {'call': 'get_robust_list',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'mq_timedsend',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'ptrace',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'sched_getattr',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'getrusage',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'setitimer',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'ioprio_get',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'vfork',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'prctl',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'move_pages',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'rt_sigreturn',
                 'reason': set([('thread_struct', 'uaccess_err')])},
                {'call': 'setpriority',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'clone',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])},
                {'call': 'sched_getparam',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('thread_struct', 'uaccess_err')])}],
 'rt_sigtimedwait': [{'call': 'keyctl',
                      'reason': set([('mm_segment_t', 'seg'),
                                     ('task_struct', 'timer_slack_ns')])},
                     {'call': 'rt_sigtimedwait',
                      'reason': set([('mm_segment_t', 'seg'),
                                     ('siginfo', 'si_code'),
                                     ('siginfo', 'si_signo'),
                                     ('task_struct', 'timer_slack_ns')])},
                     {'call': 'msgrcv',
                      'reason': set([('mm_segment_t', 'seg'),
                                     ('task_struct', 'timer_slack_ns')])},
                     {'call': 'kill',
                      'reason': set([('mm_segment_t', 'seg'),
                                     ('siginfo', 'si_code'),
                                     ('siginfo', 'si_signo'),
                                     ('task_struct', 'timer_slack_ns')])},
                     {'call': 'sched_getaffinity',
                      'reason': set([('mm_segment_t', 'seg'),
                                     ('task_struct', 'timer_slack_ns')])},
```



```
                {'call': 'sched_setparam',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'ioprio_set',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'getppid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'ioperm',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'capget',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'signal',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'semtimedop',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'umount',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'timer_create',
                 'reason': set([('siginfo', 'si_code'),
                                ('siginfo', 'si_signo')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'rt_sigqueueinfo',
                 'reason': set([('siginfo', 'si_code'),
                                ('siginfo', 'si_signo')])},
                {'call': 'tgkill',
                 'reason': set([('siginfo', 'si_code'),
                                ('siginfo', 'si_signo')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'setsid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'sigaltstack',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_setattr',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'migrate_pages',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'getitimer',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'setpgid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'getsid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'prlimit64',
```

```
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'perf_event_open',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'rt_sigaction',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'getpgid',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'getpriority',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'sigaction',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'rt_tgsigqueueinfo',
                     'reason': set([('siginfo', 'si_code'),
                                    ('siginfo', 'si_signo')])},
                    {'call': 'setns',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'fork',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'get_robust_list',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'ptrace',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'sched_getattr',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'getrusage',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'setitimer',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'ioprio_get',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'vfork',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'prctl',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'move_pages',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'rt_sigreturn',
                     'reason': set([('siginfo', 'si_code'),
                                    ('siginfo', 'si_signo')])},
                    {'call': 'tkill',
```



```
                     'reason': set([('siginfo', 'si_code'),
                                    ('siginfo', 'si_signo')])},
                    {'call': 'setpriority',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'clone',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'sched_getparam',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])}],
 'rt_tgsigqueueinfo': [{'call': 'rt_sigtimedwait',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'kill',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'timer_create',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'rt_sigqueueinfo',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'tgkill',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'rt_tgsigqueueinfo',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'rt_sigreturn',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'tkill',
                        'reason': set([('siginfo', 'si_code')])}],
 'sched_getattr': [{'call': 'keyctl',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'msgrcv',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'kill',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_setparam',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'ioprio_set',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'getppid',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'ioperm',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'capget',
```

```
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'signal',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'semtimedop',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'umount',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'setsid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sigaltstack',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sched_setattr',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('sched_attr', 'size'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'migrate_pages',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'getitimer',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'setpgid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'getsid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'prlimit64',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'perf_event_open',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'rt_sigaction',
                 'reason': set([('mm_segment_t', 'seg'),
```



```
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'getpgid',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'getpriority',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sigaction',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'setns',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'fork',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'get_robust_list',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'mq_timedsend',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'ptrace',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sched_getattr',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('sched_attr', 'size'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'getrusage',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'setitimer',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'ioprio_get',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'vfork',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
                                ('task_struct', 'sched_reset_on_fork')])},
                {'call': 'prctl',
                 'reason': set([('mm_segment_t', 'seg'),
                                ('task_struct', 'policy'),
```

```
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'move_pages',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'setpriority',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'clone',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'sched_getparam',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])}],
 'sched_getparam': [{'call': 'keyctl',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigtimedwait',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'msgrcv',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'kill',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getaffinity',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setparam',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'ioprio_set',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getppid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'mq_timedreceive',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'capget',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setaffinity',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'signal',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'semtimedop',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'umount',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_rr_get_interval',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigprocmask',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setsid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sigaltstack',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setattr',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'migrate_pages',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getitimer',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setpgid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getsid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'prlimit64',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'perf_event_open',
```



```
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigaction',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getpgid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getpriority',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sigaction',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setns',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'fork',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'get_robust_list',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'ptrace',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getattr',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getrusage',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setitimer',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'ioprio_get',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'vfork',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'prctl',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'move_pages',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setpriority',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'clone',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getparam',
                     'reason': set([('task_struct', 'policy')])}],
 'sched_getscheduler': [{'call': 'keyctl',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'rt_sigtimedwait',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'msgrcv',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'kill',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_getaffinity',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_setparam',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'ioprio_set',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'getppid',
                         'reason': set([('task_struct',
```

```
                                        'sched_reset_on_fork')])},
                          {'call': 'mq_timedreceive',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'capget',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sched_setaffinity',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'signal',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'semtimedop',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'umount',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sched_rr_get_interval',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'rt_sigprocmask',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'setsid',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sigaltstack',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sched_setattr',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'migrate_pages',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'getitimer',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'setpgid',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'getsid',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'prlimit64',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'perf_event_open',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'rt_sigaction',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'getpgid',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'getpriority',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sigaction',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'setns',
                           'reason': set([('task_struct',
```



```
                                        'sched_reset_on_fork')])},
                          {'call': 'fork',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'get_robust_list',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'mq_timedsend',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sched_getscheduler',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'ptrace',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sched_getattr',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'getrusage',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sched_setscheduler',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'setitimer',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'ioprio_get',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'vfork',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'prctl',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'move_pages',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'setpriority',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'clone',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])},
                          {'call': 'sched_getparam',
                           'reason': set([('task_struct',
                                           'sched_reset_on_fork')])}],
 'sched_rr_get_interval': [{'call': 'sched_rr_get_interval',
                            'reason': set([('rq_flags', 'flags')])}],
 'sched_setaffinity': [{'call': 'keyctl',
                        'reason': set([('cred', 'user_ns'),
                                       ('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'rt_sigtimedwait',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'setfsuid',
                        'reason': set([('cred', 'user_ns')])},
                       {'call': 'msgrcv',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'kill',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
```

```
17418                     {'call': 'getresuid16',
17419                      'reason': set([('cred', 'user_ns')])},
17420                     {'call': 'getresgid',
17421                      'reason': set([('cred', 'user_ns')])},
17422                     {'call': 'sched_getaffinity',
17423                      'reason': set([('task_struct', 'flags'),
17424                                     ('task_struct', 'real_cred')])},
17425                     {'call': 'sched_setparam',
17426                      'reason': set([('task_struct', 'flags'),
17427                                     ('task_struct', 'real_cred')])},
17428                     {'call': 'setgid',
17429                      'reason': set([('cred', 'user_ns')])},
17430                     {'call': 'ioprio_set',
17431                      'reason': set([('cred', 'user_ns'),
17432                                     ('task_struct', 'flags'),
17433                                     ('task_struct', 'real_cred')])},
17434                     {'call': 'capset',
17435                      'reason': set([('cred', 'user_ns')])},
17436                     {'call': 'getppid',
17437                      'reason': set([('task_struct', 'flags'),
17438                                     ('task_struct', 'real_cred')])},
17439                     {'call': 'mq_timedreceive',
17440                      'reason': set([('task_struct', 'flags'),
17441                                     ('task_struct', 'real_cred')])},
17442                     {'call': 'getresgid16',
17443                      'reason': set([('cred', 'user_ns')])},
17444                     {'call': 'capget',
17445                      'reason': set([('task_struct', 'flags'),
17446                                     ('task_struct', 'real_cred')])},
17447                     {'call': 'sched_setaffinity',
17448                      'reason': set([('cred', 'user_ns'),
17449                                     ('task_struct', 'flags'),
17450                                     ('task_struct', 'real_cred')])},
17451                     {'call': 'setfsgid',
17452                      'reason': set([('cred', 'user_ns')])},
17453                     {'call': 'unshare',
17454                      'reason': set([('cred', 'user_ns')])},
17455                     {'call': 'signal',
17456                      'reason': set([('task_struct', 'flags'),
17457                                     ('task_struct', 'real_cred')])},
17458                     {'call': 'setreuid',
17459                      'reason': set([('cred', 'user_ns'),
17460                                     ('task_struct', 'flags')])},
17461                     {'call': 'semtimedop',
17462                      'reason': set([('task_struct', 'flags'),
17463                                     ('task_struct', 'real_cred')])},
17464                     {'call': 'umount',
17465                      'reason': set([('task_struct', 'flags'),
17466                                     ('task_struct', 'real_cred')])},
17467                     {'call': 'sched_rr_get_interval',
17468                      'reason': set([('task_struct', 'flags'),
17469                                     ('task_struct', 'real_cred')])},
17470                     {'call': 'epoll_create1',
17471                      'reason': set([('cred', 'user_ns')])},
17472                     {'call': 'getresuid',
17473                      'reason': set([('cred', 'user_ns')])},
17474                     {'call': 'rt_sigprocmask',
17475                      'reason': set([('task_struct', 'flags'),
17476                                     ('task_struct', 'real_cred')])},
17477                     {'call': 'setsid',
17478                      'reason': set([('task_struct', 'flags'),
17479                                     ('task_struct', 'real_cred')])},
17480                     {'call': 'sigaltstack',
17481                      'reason': set([('task_struct', 'flags'),
17482                                     ('task_struct', 'real_cred')])},
17483                     {'call': 'sched_setattr',
```



```
17484                      'reason': set([('task_struct', 'flags'),
17485                                     ('task_struct', 'real_cred')])},
17486                     {'call': 'migrate_pages',
17487                      'reason': set([('cred', 'user_ns'),
17488                                     ('task_struct', 'flags'),
17489                                     ('task_struct', 'real_cred')])},
17490                     {'call': 'getitimer',
17491                      'reason': set([('task_struct', 'flags'),
17492                                     ('task_struct', 'real_cred')])},
17493                     {'call': 'setpgid',
17494                      'reason': set([('task_struct', 'flags'),
17495                                     ('task_struct', 'real_cred')])},
17496                     {'call': 'setresgid',
17497                      'reason': set([('cred', 'user_ns')])},
17498                     {'call': 'setregid',
17499                      'reason': set([('cred', 'user_ns')])},
17500                     {'call': 'getsid',
17501                      'reason': set([('task_struct', 'flags'),
17502                                     ('task_struct', 'real_cred')])},
17503                     {'call': 'prlimit64',
17504                      'reason': set([('cred', 'user_ns'),
17505                                     ('task_struct', 'flags'),
17506                                     ('task_struct', 'real_cred')])},
17507                     {'call': 'perf_event_open',
17508                      'reason': set([('task_struct', 'flags'),
17509                                     ('task_struct', 'real_cred')])},
17510                     {'call': 'getgroups16',
17511                      'reason': set([('cred', 'user_ns')])},
17512                     {'call': 'rt_sigaction',
17513                      'reason': set([('task_struct', 'flags'),
17514                                     ('task_struct', 'real_cred')])},
17515                     {'call': 'getpgid',
17516                      'reason': set([('task_struct', 'flags'),
17517                                     ('task_struct', 'real_cred')])},
17518                     {'call': 'getpriority',
17519                      'reason': set([('cred', 'user_ns'),
17520                                     ('task_struct', 'flags'),
17521                                     ('task_struct', 'real_cred')])},
17522                     {'call': 'sigaction',
17523                      'reason': set([('task_struct', 'flags'),
17524                                     ('task_struct', 'real_cred')])},
17525                     {'call': 'faccessat',
17526                      'reason': set([('cred', 'user_ns')])},
17527                     {'call': 'setns',
17528                      'reason': set([('task_struct', 'flags'),
17529                                     ('task_struct', 'real_cred')])},
17530                     {'call': 'fork',
17531                      'reason': set([('task_struct', 'flags'),
17532                                     ('task_struct', 'real_cred')])},
17533                     {'call': 'get_robust_list',
17534                      'reason': set([('task_struct', 'flags'),
17535                                     ('task_struct', 'real_cred')])},
17536                     {'call': 'mq_timedsend',
17537                      'reason': set([('task_struct', 'flags'),
17538                                     ('task_struct', 'real_cred')])},
17539                     {'call': 'sched_getscheduler',
17540                      'reason': set([('task_struct', 'flags'),
17541                                     ('task_struct', 'real_cred')])},
17542                     {'call': 'ptrace',
17543                      'reason': set([('task_struct', 'flags'),
17544                                     ('task_struct', 'real_cred')])},
17545                     {'call': 'sched_getattr',
17546                      'reason': set([('task_struct', 'flags'),
17547                                     ('task_struct', 'real_cred')])},
17548                     {'call': 'getrusage',
17549                      'reason': set([('task_struct', 'flags'),
```

```
17550                                       ('task_struct', 'real_cred')])},
17551                           {'call': 'sched_setscheduler',
17552                            'reason': set([('task_struct', 'flags'),
17553                                           ('task_struct', 'real_cred')])},
17554                           {'call': 'setresuid',
17555                            'reason': set([('cred', 'user_ns'),
17556                                           ('task_struct', 'flags')])},
17557                           {'call': 'setitimer',
17558                            'reason': set([('task_struct', 'flags'),
17559                                           ('task_struct', 'real_cred')])},
17560                           {'call': 'ioprio_get',
17561                            'reason': set([('cred', 'user_ns'),
17562                                           ('task_struct', 'flags'),
17563                                           ('task_struct', 'real_cred')])},
17564                           {'call': 'vfork',
17565                            'reason': set([('task_struct', 'flags'),
17566                                           ('task_struct', 'real_cred')])},
17567                           {'call': 'setuid',
17568                            'reason': set([('cred', 'user_ns'),
17569                                           ('task_struct', 'flags')])},
17570                           {'call': 'prctl',
17571                            'reason': set([('task_struct', 'flags'),
17572                                           ('task_struct', 'real_cred')])},
17573                           {'call': 'move_pages',
17574                            'reason': set([('task_struct', 'flags'),
17575                                           ('task_struct', 'real_cred')])},
17576                           {'call': 'getgroups',
17577                            'reason': set([('cred', 'user_ns')])},
17578                           {'call': 'setpriority',
17579                            'reason': set([('cred', 'user_ns'),
17580                                           ('task_struct', 'flags'),
17581                                           ('task_struct', 'real_cred')])},
17582                           {'call': 'clone',
17583                            'reason': set([('task_struct', 'flags'),
17584                                           ('task_struct', 'real_cred')])},
17585                           {'call': 'sched_getparam',
17586                            'reason': set([('task_struct', 'flags'),
17587                                           ('task_struct', 'real_cred')])}],
17588  'sched_setattr': [{'call': 'keyctl',
17589                     'reason': set([('mm_segment_t', 'seg')])},
17590                    {'call': 'rt_sigtimedwait',
17591                     'reason': set([('mm_segment_t', 'seg')])},
17592                    {'call': 'msgrcv',
17593                     'reason': set([('mm_segment_t', 'seg')])},
17594                    {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
17595                    {'call': 'sched_getaffinity',
17596                     'reason': set([('mm_segment_t', 'seg')])},
17597                    {'call': 'sched_setparam',
17598                     'reason': set([('mm_segment_t', 'seg')])},
17599                    {'call': 'ioprio_set',
17600                     'reason': set([('mm_segment_t', 'seg')])},
17601                    {'call': 'getppid',
17602                     'reason': set([('mm_segment_t', 'seg')])},
17603                    {'call': 'ioperm',
17604                     'reason': set([('mm_segment_t', 'seg')])},
17605                    {'call': 'mq_timedreceive',
17606                     'reason': set([('mm_segment_t', 'seg')])},
17607                    {'call': 'capget',
17608                     'reason': set([('mm_segment_t', 'seg')])},
17609                    {'call': 'sched_setaffinity',
17610                     'reason': set([('mm_segment_t', 'seg')])},
17611                    {'call': 'signal',
17612                     'reason': set([('mm_segment_t', 'seg')])},
17613                    {'call': 'semtimedop',
17614                     'reason': set([('mm_segment_t', 'seg')])},
17615                    {'call': 'umount',
```



```
17616                     'reason': set([('mm_segment_t', 'seg')])},
17617                    {'call': 'sched_rr_get_interval',
17618                     'reason': set([('mm_segment_t', 'seg')])},
17619                    {'call': 'rt_sigprocmask',
17620                     'reason': set([('mm_segment_t', 'seg')])},
17621                    {'call': 'setsid',
17622                     'reason': set([('mm_segment_t', 'seg')])},
17623                    {'call': 'sigaltstack',
17624                     'reason': set([('mm_segment_t', 'seg')])},
17625                    {'call': 'sched_setattr',
17626                     'reason': set([('mm_segment_t', 'seg'),
17627                                    ('sched_attr', 'sched_policy')])},
17628                    {'call': 'migrate_pages',
17629                     'reason': set([('mm_segment_t', 'seg')])},
17630                    {'call': 'getitimer',
17631                     'reason': set([('mm_segment_t', 'seg')])},
17632                    {'call': 'setpgid',
17633                     'reason': set([('mm_segment_t', 'seg')])},
17634                    {'call': 'getsid',
17635                     'reason': set([('mm_segment_t', 'seg')])},
17636                    {'call': 'prlimit64',
17637                     'reason': set([('mm_segment_t', 'seg')])},
17638                    {'call': 'perf_event_open',
17639                     'reason': set([('mm_segment_t', 'seg')])},
17640                    {'call': 'rt_sigaction',
17641                     'reason': set([('mm_segment_t', 'seg')])},
17642                    {'call': 'getpgid',
17643                     'reason': set([('mm_segment_t', 'seg')])},
17644                    {'call': 'getpriority',
17645                     'reason': set([('mm_segment_t', 'seg')])},
17646                    {'call': 'sigaction',
17647                     'reason': set([('mm_segment_t', 'seg')])},
17648                    {'call': 'setns',
17649                     'reason': set([('mm_segment_t', 'seg')])},
17650                    {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
17651                    {'call': 'get_robust_list',
17652                     'reason': set([('mm_segment_t', 'seg')])},
17653                    {'call': 'mq_timedsend',
17654                     'reason': set([('mm_segment_t', 'seg')])},
17655                    {'call': 'sched_getscheduler',
17656                     'reason': set([('mm_segment_t', 'seg')])},
17657                    {'call': 'ptrace',
17658                     'reason': set([('mm_segment_t', 'seg')])},
17659                    {'call': 'sched_getattr',
17660                     'reason': set([('mm_segment_t', 'seg'),
17661                                    ('sched_attr', 'sched_policy')])},
17662                    {'call': 'getrusage',
17663                     'reason': set([('mm_segment_t', 'seg')])},
17664                    {'call': 'sched_setscheduler',
17665                     'reason': set([('mm_segment_t', 'seg')])},
17666                    {'call': 'setitimer',
17667                     'reason': set([('mm_segment_t', 'seg')])},
17668                    {'call': 'ioprio_get',
17669                     'reason': set([('mm_segment_t', 'seg')])},
17670                    {'call': 'vfork',
17671                     'reason': set([('mm_segment_t', 'seg')])},
17672                    {'call': 'prctl',
17673                     'reason': set([('mm_segment_t', 'seg')])},
17674                    {'call': 'move_pages',
17675                     'reason': set([('mm_segment_t', 'seg')])},
17676                    {'call': 'setpriority',
17677                     'reason': set([('mm_segment_t', 'seg')])},
17678                    {'call': 'clone',
17679                     'reason': set([('mm_segment_t', 'seg')])},
17680                    {'call': 'sched_getparam',
17681                     'reason': set([('mm_segment_t', 'seg')])}],
```

```
'select': [{'call': 'keyctl',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('task_struct', 'personality'),
                           ('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'msgrcv',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'mq_unlink',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
           {'call': 'swapoff',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'sched_getaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setparam',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'fchmod',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'memfd_create',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'ioprio_set',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'personality',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'dup3', 'reason': set([('files_struct', 'fdt')])},
           {'call': 'readlinkat',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'io_getevents',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'getppid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'fchown',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'mq_timedreceive',
            'reason': set([('task_struct', 'personality'),
                           ('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'capget',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'utime',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'sched_setaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'unshare', 'reason': set([('files_struct', 'fdt')])},
           {'call': 'signal',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'semtimedop',
            'reason': set([('task_struct', 'personality'),
                           ('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'umount',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'settimeofday',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'sched_rr_get_interval',
            'reason': set([('task_struct', 'personality'),
```



```
                           ('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'timerfd_gettime',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'pselect6',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'uselib',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'rt_sigprocmask',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'setsid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sigaltstack',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setattr',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'migrate_pages',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getitimer',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'fchmodat',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'setpgid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'inotify_add_watch',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'timer_settime',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'ftruncate',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'timer_gettime',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'getsid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'ioctl',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'prlimit64',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'perf_event_open',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'linkat',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'stime',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'rt_sigaction',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'futimesat',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'getpgid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'poll',
            'reason': set([('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'select',
```

```
17814              'reason': set([('fd_set_bits', 'ex'),
17815                             ('fd_set_bits', 'in'),
17816                             ('fd_set_bits', 'out'),
17817                             ('fd_set_bits', 'res_ex'),
17818                             ('fd_set_bits', 'res_in'),
17819                             ('fd_set_bits', 'res_out'),
17820                             ('timespec', 'tv_nsec'),
17821                             ('timespec', 'tv_sec')])},
17822             {'call': 'unlink',
17823              'reason': set([('timespec', 'tv_nsec'),
17824                             ('timespec', 'tv_sec')])},
17825             {'call': 'getpriority',
17826              'reason': set([('task_struct', 'personality')])},
17827             {'call': 'sigaction',
17828              'reason': set([('task_struct', 'personality')])},
17829             {'call': 'nanosleep',
17830              'reason': set([('timespec', 'tv_nsec'),
17831                             ('timespec', 'tv_sec')])},
17832             {'call': 'mq_getsetattr',
17833              'reason': set([('timespec', 'tv_nsec'),
17834                             ('timespec', 'tv_sec')])},
17835             {'call': 'faccessat',
17836              'reason': set([('timespec', 'tv_nsec'),
17837                             ('timespec', 'tv_sec')])},
17838             {'call': 'setns',
17839              'reason': set([('task_struct', 'personality')])},
17840             {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
17841             {'call': 'get_robust_list',
17842              'reason': set([('task_struct', 'personality')])},
17843             {'call': 'mq_timedsend',
17844              'reason': set([('task_struct', 'personality'),
17845                             ('timespec', 'tv_nsec'),
17846                             ('timespec', 'tv_sec')])},
17847             {'call': 'sched_getscheduler',
17848              'reason': set([('task_struct', 'personality')])},
17849             {'call': 'ptrace',
17850              'reason': set([('task_struct', 'personality')])},
17851             {'call': 'swapon',
17852              'reason': set([('timespec', 'tv_nsec'),
17853                             ('timespec', 'tv_sec')])},
17854             {'call': 'epoll_wait',
17855              'reason': set([('timespec', 'tv_nsec'),
17856                             ('timespec', 'tv_sec')])},
17857             {'call': 'sched_getattr',
17858              'reason': set([('task_struct', 'personality')])},
17859             {'call': 'fchownat',
17860              'reason': set([('timespec', 'tv_nsec'),
17861                             ('timespec', 'tv_sec')])},
17862             {'call': 'getrusage',
17863              'reason': set([('task_struct', 'personality')])},
17864             {'call': 'fstat',
17865              'reason': set([('timespec', 'tv_nsec'),
17866                             ('timespec', 'tv_sec')])},
17867             {'call': 'timerfd_settime',
17868              'reason': set([('timespec', 'tv_nsec'),
17869                             ('timespec', 'tv_sec')])},
17870             {'call': 'sched_setscheduler',
17871              'reason': set([('task_struct', 'personality')])},
17872             {'call': 'setitimer',
17873              'reason': set([('task_struct', 'personality')])},
17874             {'call': 'ioprio_get',
17875              'reason': set([('task_struct', 'personality')])},
17876             {'call': 'vfork',
17877              'reason': set([('task_struct', 'personality')])},
17878             {'call': 'prctl',
17879              'reason': set([('task_struct', 'personality')])},
```



```
17880             {'call': 'move_pages',
17881              'reason': set([('task_struct', 'personality')])},
17882             {'call': 'dup2', 'reason': set([('files_struct', 'fdt')])},
17883             {'call': 'setpriority',
17884              'reason': set([('task_struct', 'personality')])},
17885             {'call': 'mq_notify',
17886              'reason': set([('timespec', 'tv_nsec'),
17887                             ('timespec', 'tv_sec')])},
17888             {'call': 'sendfile',
17889              'reason': set([('timespec', 'tv_nsec'),
17890                             ('timespec', 'tv_sec')])},
17891             {'call': 'newfstat',
17892              'reason': set([('timespec', 'tv_nsec'),
17893                             ('timespec', 'tv_sec')])},
17894             {'call': 'clone',
17895              'reason': set([('task_struct', 'personality')])},
17896             {'call': 'clock_nanosleep',
17897              'reason': set([('timespec', 'tv_nsec'),
17898                             ('timespec', 'tv_sec')])},
17899             {'call': 'unlinkat',
17900              'reason': set([('timespec', 'tv_nsec'),
17901                             ('timespec', 'tv_sec')])},
17902             {'call': 'sched_getparam',
17903              'reason': set([('task_struct', 'personality')])},
17904             {'call': 'futex',
17905              'reason': set([('timespec', 'tv_nsec'),
17906                             ('timespec', 'tv_sec')])},
17907             {'call': 'recvmmsg',
17908              'reason': set([('timespec', 'tv_nsec'),
17909                             ('timespec', 'tv_sec')])},
17910             {'call': 'sendfile64',
17911              'reason': set([('timespec', 'tv_nsec'),
17912                             ('timespec', 'tv_sec')])},
17913             {'call': 'ppoll',
17914              'reason': set([('timespec', 'tv_nsec'),
17915                             ('timespec', 'tv_sec')])}],
17916  'semctl': [{'call': 'semtimedop',
17917              'reason': set([('sem_array', 'sem_nsems')])},
17918             {'call': 'semctl', 'reason': set([('sem_array', 'sem_nsems')])}],
17919  'semtimedop': [{'call': 'syncfs', 'reason': set([('list_head', 'prev')])},
17920                 {'call': 'keyctl', 'reason': set([('list_head', 'prev')])},
17921                 {'call': 'rt_sigtimedwait',
17922                  'reason': set([('list_head', 'prev'),
17923                                 ('timespec', 'tv_nsec'),
17924                                 ('timespec', 'tv_sec')])},
17925                 {'call': 'vmsplice', 'reason': set([('list_head', 'prev')])},
17926                 {'call': 'msgrcv',
17927                  'reason': set([('kern_ipc_perm', 'deleted'),
17928                                 ('list_head', 'prev')])},
17929                 {'call': 'eventfd2', 'reason': set([('list_head', 'prev')])},
17930                 {'call': 'mq_unlink',
17931                  'reason': set([('list_head', 'prev'),
17932                                 ('timespec', 'tv_nsec'),
17933                                 ('timespec', 'tv_sec')])},
17934                 {'call': 'kill', 'reason': set([('list_head', 'prev')])},
17935                 {'call': 'swapoff',
17936                  'reason': set([('list_head', 'prev'),
17937                                 ('timespec', 'tv_nsec'),
17938                                 ('timespec', 'tv_sec')])},
17939                 {'call': 'readahead', 'reason': set([('list_head', 'prev')])},
17940                 {'call': 'timer_delete',
17941                  'reason': set([('list_head', 'prev')])},
17942                 {'call': 'sched_getaffinity',
17943                  'reason': set([('list_head', 'prev')])},
17944                 {'call': 'sched_setparam',
17945                  'reason': set([('list_head', 'prev')])},
```

```
17946                 {'call': 'fchmod',
17947                  'reason': set([('list_head', 'prev'),
17948                                 ('timespec', 'tv_nsec'),
17949                                 ('timespec', 'tv_sec')])},
17950                 {'call': 'setgid', 'reason': set([('list_head', 'prev')])},
17951                 {'call': 'pivot_root',
17952                  'reason': set([('list_head', 'prev')])},
17953                 {'call': 'memfd_create',
17954                  'reason': set([('list_head', 'prev'),
17955                                 ('timespec', 'tv_nsec'),
17956                                 ('timespec', 'tv_sec')])},
17957                 {'call': 'ioprio_set',
17958                  'reason': set([('list_head', 'prev')])},
17959                 {'call': 'delete_module',
17960                  'reason': set([('list_head', 'prev')])},
17961                 {'call': 'remap_file_pages',
17962                  'reason': set([('list_head', 'prev')])},
17963                 {'call': 'dup3', 'reason': set([('list_head', 'prev')])},
17964                 {'call': 'readlinkat',
17965                  'reason': set([('list_head', 'prev'),
17966                                 ('timespec', 'tv_nsec'),
17967                                 ('timespec', 'tv_sec')])},
17968                 {'call': 'io_getevents',
17969                  'reason': set([('list_head', 'prev'),
17970                                 ('timespec', 'tv_nsec'),
17971                                 ('timespec', 'tv_sec')])},
17972                 {'call': 'getppid', 'reason': set([('list_head', 'prev')])},
17973                 {'call': 'fchown',
17974                  'reason': set([('list_head', 'prev'),
17975                                 ('timespec', 'tv_nsec'),
17976                                 ('timespec', 'tv_sec')])},
17977                 {'call': 'mq_timedreceive',
17978                  'reason': set([('list_head', 'prev'),
17979                                 ('timespec', 'tv_nsec'),
17980                                 ('timespec', 'tv_sec')])},
17981                 {'call': 'capget', 'reason': set([('list_head', 'prev')])},
17982                 {'call': 'utime',
17983                  'reason': set([('timespec', 'tv_nsec'),
17984                                 ('timespec', 'tv_sec')])},
17985                 {'call': 'sched_setaffinity',
17986                  'reason': set([('list_head', 'prev')])},
17987                 {'call': 'ustat', 'reason': set([('list_head', 'prev')])},
17988                 {'call': 'bpf', 'reason': set([('list_head', 'prev')])},
17989                 {'call': 'unshare', 'reason': set([('list_head', 'prev')])},
17990                 {'call': 'signal', 'reason': set([('list_head', 'prev')])},
17991                 {'call': 'setreuid', 'reason': set([('list_head', 'prev')])},
17992                 {'call': 'semtimedop',
17993                  'reason': set([('kern_ipc_perm', 'deleted'),
17994                                 ('list_head', 'prev'),
17995                                 ('sem_array', 'complex_count'),
17996                                 ('sem_array', 'sem_nsems'),
17997                                 ('sem_array', 'use_global_lock'),
17998                                 ('sem_queue', 'dupsop'),
17999                                 ('sem_queue', 'nsops'),
18000                                 ('sem_undo', 'semid'),
18001                                 ('sembuf', 'sem_flg'),
18002                                 ('sembuf', 'sem_num'),
18003                                 ('sembuf', 'sem_op'),
18004                                 ('timespec', 'tv_nsec'),
18005                                 ('timespec', 'tv_sec')])},
18006                 {'call': 'umount', 'reason': set([('list_head', 'prev')])},
18007                 {'call': 'settimeofday',
18008                  'reason': set([('timespec', 'tv_nsec'),
18009                                 ('timespec', 'tv_sec')])},
18010                 {'call': 'timer_create',
18011                  'reason': set([('list_head', 'prev')])},
```



```
18012                 {'call': 'mkdirat', 'reason': set([('list_head', 'prev')])},
18013                 {'call': 'sched_rr_get_interval',
18014                  'reason': set([('list_head', 'prev'),
18015                                 ('timespec', 'tv_nsec'),
18016                                 ('timespec', 'tv_sec')])},
18017                 {'call': 'epoll_create1',
18018                  'reason': set([('list_head', 'prev')])},
18019                 {'call': 'timerfd_gettime',
18020                  'reason': set([('list_head', 'prev'),
18021                                 ('timespec', 'tv_nsec'),
18022                                 ('timespec', 'tv_sec')])},
18023                 {'call': 'tee', 'reason': set([('list_head', 'prev')])},
18024                 {'call': 'semctl',
18025                  'reason': set([('kern_ipc_perm', 'deleted'),
18026                                 ('list_head', 'prev'),
18027                                 ('sem_array', 'complex_count'),
18028                                 ('sem_array', 'sem_nsems'),
18029                                 ('sem_array', 'use_global_lock'),
18030                                 ('sem_undo', 'semid')])},
18031                 {'call': 'sync_file_range',
18032                  'reason': set([('list_head', 'prev')])},
18033                 {'call': 'epoll_ctl', 'reason': set([('list_head', 'prev')])},
18034                 {'call': 'flock', 'reason': set([('list_head', 'prev')])},
18035                 {'call': 'openat', 'reason': set([('list_head', 'prev')])},
18036                 {'call': 'lookup_dcookie',
18037                  'reason': set([('list_head', 'prev')])},
18038                 {'call': 'pselect6',
18039                  'reason': set([('timespec', 'tv_nsec'),
18040                                 ('timespec', 'tv_sec')])},
18041                 {'call': 'uselib',
18042                  'reason': set([('list_head', 'prev'),
18043                                 ('timespec', 'tv_nsec'),
18044                                 ('timespec', 'tv_sec')])},
18045                 {'call': 'renameat2', 'reason': set([('list_head', 'prev')])},
18046                 {'call': 'rt_sigprocmask',
18047                  'reason': set([('list_head', 'prev')])},
18048                 {'call': 'accept4', 'reason': set([('list_head', 'prev')])},
18049                 {'call': 'msgctl',
18050                  'reason': set([('kern_ipc_perm', 'deleted'),
18051                                 ('list_head', 'prev')])},
18052                 {'call': 'reboot', 'reason': set([('list_head', 'prev')])},
18053                 {'call': 'setsid', 'reason': set([('list_head', 'prev')])},
18054                 {'call': 'set_trip_temp',
18055                  'reason': set([('list_head', 'prev')])},
18056                 {'call': 'sigaltstack',
18057                  'reason': set([('list_head', 'prev')])},
18058                 {'call': 'sched_setattr',
18059                  'reason': set([('list_head', 'prev')])},
18060                 {'call': 'inotify_rm_watch',
18061                  'reason': set([('list_head', 'prev')])},
18062                 {'call': 'socketpair',
18063                  'reason': set([('list_head', 'prev')])},
18064                 {'call': 'migrate_pages',
18065                  'reason': set([('list_head', 'prev')])},
18066                 {'call': 'getitimer', 'reason': set([('list_head', 'prev')])},
18067                 {'call': 'fchmodat',
18068                  'reason': set([('list_head', 'prev'),
18069                                 ('timespec', 'tv_nsec'),
18070                                 ('timespec', 'tv_sec')])},
18071                 {'call': 'setpgid', 'reason': set([('list_head', 'prev')])},
18072                 {'call': 'init_module',
18073                  'reason': set([('list_head', 'prev')])},
18074                 {'call': 'setresgid', 'reason': set([('list_head', 'prev')])},
18075                 {'call': 'getcwd', 'reason': set([('list_head', 'prev')])},
18076                 {'call': 'inotify_add_watch',
18077                  'reason': set([('list_head', 'prev'),
```

```
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'get_trip_temp',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'timer_settime',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'setregid', 'reason': set([('list_head', 'prev')])},
                {'call': 'splice', 'reason': set([('list_head', 'prev')])},
                {'call': 'ftruncate',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'timer_gettime',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'getsid', 'reason': set([('list_head', 'prev')])},
                {'call': 'shmat',
                 'reason': set([('kern_ipc_perm', 'deleted'),
                                ('list_head', 'prev')])},
                {'call': 'mknodat', 'reason': set([('list_head', 'prev')])},
                {'call': 'socket', 'reason': set([('list_head', 'prev')])},
                {'call': 'symlinkat', 'reason': set([('list_head', 'prev')])},
                {'call': 'pipe2', 'reason': set([('list_head', 'prev')])},
                {'call': 'ioctl',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'prlimit64', 'reason': set([('list_head', 'prev')])},
                {'call': 'perf_event_open',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'linkat',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'stime',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'getgroups16',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'shmdt', 'reason': set([('list_head', 'prev')])},
                {'call': 'quotactl', 'reason': set([('list_head', 'prev')])},
                {'call': 'rt_sigaction',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'futimesat',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'request_key',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getpgid', 'reason': set([('list_head', 'prev')])},
                {'call': 'brk', 'reason': set([('list_head', 'prev')])},
                {'call': 'acct', 'reason': set([('list_head', 'prev')])},
                {'call': 'poll',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'open', 'reason': set([('list_head', 'prev')])},
                {'call': 'select',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'unlink',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'exit_group',
```



```
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getpriority',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'sigaction', 'reason': set([('list_head', 'prev')])},
                {'call': 'nanosleep',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'mq_getsetattr',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'faccessat',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'rmdir', 'reason': set([('list_head', 'prev')])},
                {'call': 'dup', 'reason': set([('list_head', 'prev')])},
                {'call': 'setgroups16',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'setns', 'reason': set([('list_head', 'prev')])},
                {'call': 'fork', 'reason': set([('list_head', 'prev')])},
                {'call': 'get_mempolicy',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'io_submit', 'reason': set([('list_head', 'prev')])},
                {'call': 'get_robust_list',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'mq_timedsend',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'sched_yield',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'ptrace', 'reason': set([('list_head', 'prev')])},
                {'call': 'shmctl',
                 'reason': set([('kern_ipc_perm', 'deleted'),
                                ('list_head', 'prev')])},
                {'call': 'munlockall',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'swapon',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'pkey_mprotect',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'madvise', 'reason': set([('list_head', 'prev')])},
                {'call': 'epoll_wait',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'sched_getattr',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'fchownat',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'getrusage', 'reason': set([('list_head', 'prev')])},
                {'call': 'fstat',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'timerfd_settime',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'sched_setscheduler',
```

```
                 'reason': set([('list_head', 'prev')])},
                {'call': 'setresuid', 'reason': set([('list_head', 'prev')])},
                {'call': 'setitimer', 'reason': set([('list_head', 'prev')])},
                {'call': 'ioprio_get',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'vfork', 'reason': set([('list_head', 'prev')])},
                {'call': 'setuid', 'reason': set([('list_head', 'prev')])},
                {'call': 'io_setup', 'reason': set([('list_head', 'prev')])},
                {'call': 'mprotect', 'reason': set([('list_head', 'prev')])},
                {'call': 'mmap_pgoff',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'mremap', 'reason': set([('list_head', 'prev')])},
                {'call': 'io_destroy',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'mbind', 'reason': set([('list_head', 'prev')])},
                {'call': 'prctl', 'reason': set([('list_head', 'prev')])},
                {'call': 'move_pages',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'timerfd_create',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'modify_ldt',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getgroups', 'reason': set([('list_head', 'prev')])},
                {'call': 'dup2', 'reason': set([('list_head', 'prev')])},
                {'call': 'get_curr_temp',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'msgsnd',
                 'reason': set([('kern_ipc_perm', 'deleted'),
                                ('list_head', 'prev')])},
                {'call': 'munlock', 'reason': set([('list_head', 'prev')])},
                {'call': 'setpriority',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'inotify_init1',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'mincore', 'reason': set([('list_head', 'prev')])},
                {'call': 'mq_notify',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'sendfile',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'timer_getoverrun',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'newfstat',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'kexec_load',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'clone', 'reason': set([('list_head', 'prev')])},
                {'call': 'mq_open', 'reason': set([('list_head', 'prev')])},
                {'call': 'setgroups', 'reason': set([('list_head', 'prev')])},
                {'call': 'clock_nanosleep',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'unlinkat',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'sched_getparam',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'io_cancel', 'reason': set([('list_head', 'prev')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'futex',
```



```
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'recvmmsg',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'finit_module',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'sendfile64',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'mlockall', 'reason': set([('list_head', 'prev')])},
                {'call': 'ppoll',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])}],
 'sendfile': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
              {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
              {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
              {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
              {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
              {'call': 'readahead', 'reason': set([('fd', 'file')])},
              {'call': 'getdents', 'reason': set([('fd', 'file')])},
              {'call': 'writev', 'reason': set([('fd', 'file')])},
              {'call': 'preadv64', 'reason': set([('fd', 'file')])},
              {'call': 'fchmod', 'reason': set([('fd', 'file')])},
              {'call': 'pread64', 'reason': set([('fd', 'file')])},
              {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
              {'call': 'remap_file_pages',
               'reason': set([('file', 'f_mode')])},
              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
              {'call': 'read', 'reason': set([('fd', 'file')])},
              {'call': 'fchown', 'reason': set([('fd', 'file')])},
              {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
              {'call': 'utime', 'reason': set([('fd', 'file')])},
              {'call': 'fsync', 'reason': set([('fd', 'file')])},
              {'call': 'bpf', 'reason': set([('fd', 'file')])},
              {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
              {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
              {'call': 'sendto', 'reason': set([('fd', 'file')])},
              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
              {'call': 'tee', 'reason': set([('fd', 'file')])},
              {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
              {'call': 'lseek', 'reason': set([('fd', 'file')])},
              {'call': 'connect', 'reason': set([('fd', 'file')])},
              {'call': 'getsockname', 'reason': set([('fd', 'file')])},
              {'call': 'epoll_ctl',
               'reason': set([('fd', 'file'), ('file', 'f_mode')])},
              {'call': 'flock',
               'reason': set([('fd', 'file'), ('file', 'f_mode')])},
              {'call': 'pwritev', 'reason': set([('fd', 'file')])},
              {'call': 'fchdir', 'reason': set([('fd', 'file')])},
              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
              {'call': 'accept4',
               'reason': set([('fd', 'file'), ('file', 'f_mode')])},
              {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
              {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
              {'call': 'utimensat', 'reason': set([('fd', 'file')])},
              {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
              {'call': 'preadv2', 'reason': set([('fd', 'file')])},
              {'call': 'splice', 'reason': set([('fd', 'file')])},
              {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
              {'call': 'preadv', 'reason': set([('fd', 'file')])},
```

```
18342                {'call': 'getpeername', 'reason': set([('fd', 'file')])},
18343                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
18344                {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
18345                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
18346                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
18347                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
18348                {'call': 'ioctl', 'reason': set([('fd', 'file')])},
18349                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
18350                {'call': 'perf_event_open',
18351                 'reason': set([('fd', 'file'), ('file', 'f_mode')])},
18352                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
18353                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
18354                {'call': 'futimesat', 'reason': set([('fd', 'file')])},
18355                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
18356                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
18357                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
18358                {'call': 'open', 'reason': set([('file', 'f_mode')])},
18359                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
18360                {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
18361                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
18362                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
18363                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
18364                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
18365                {'call': 'listen', 'reason': set([('fd', 'file')])},
18366                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
18367                {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
18368                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
18369                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
18370                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
18371                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
18372                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
18373                {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
18374                {'call': 'llseek', 'reason': set([('fd', 'file')])},
18375                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
18376                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
18377                {'call': 'readv', 'reason': set([('fd', 'file')])},
18378                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
18379                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
18380                {'call': 'write', 'reason': set([('fd', 'file')])},
18381                {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
18382                {'call': 'sendfile', 'reason': set([('fd', 'file')])},
18383                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
18384                {'call': 'open_by_handle_at',
18385                 'reason': set([('file', 'f_mode')])},
18386                {'call': 'bind', 'reason': set([('fd', 'file')])},
18387                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
18388                {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
18389  'sendfile64': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
18390                 {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
18391                 {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
18392                 {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
18393                 {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
18394                 {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
18395                 {'call': 'readahead', 'reason': set([('fd', 'file')])},
18396                 {'call': 'getdents', 'reason': set([('fd', 'file')])},
18397                 {'call': 'writev', 'reason': set([('fd', 'file')])},
18398                 {'call': 'preadv64', 'reason': set([('fd', 'file')])},
18399                 {'call': 'fchmod', 'reason': set([('fd', 'file')])},
18400                 {'call': 'pread64', 'reason': set([('fd', 'file')])},
18401                 {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
18402                 {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
18403                 {'call': 'remap_file_pages',
18404                  'reason': set([('file', 'f_mode')])},
18405                 {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
18406                 {'call': 'read', 'reason': set([('fd', 'file')])},
18407                 {'call': 'fchown', 'reason': set([('fd', 'file')])},
```



```
18408                {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
18409                {'call': 'utime', 'reason': set([('fd', 'file')])},
18410                {'call': 'fsync', 'reason': set([('fd', 'file')])},
18411                {'call': 'bpf', 'reason': set([('fd', 'file')])},
18412                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
18413                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
18414                {'call': 'sendto', 'reason': set([('fd', 'file')])},
18415                {'call': 'epoll_create1',
18416                 'reason': set([('file', 'f_mode')])},
18417                {'call': 'tee', 'reason': set([('fd', 'file')])},
18418                {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
18419                {'call': 'lseek', 'reason': set([('fd', 'file')])},
18420                {'call': 'connect', 'reason': set([('fd', 'file')])},
18421                {'call': 'getsockname', 'reason': set([('fd', 'file')])},
18422                {'call': 'epoll_ctl',
18423                 'reason': set([('fd', 'file'), ('file', 'f_mode')])},
18424                {'call': 'flock',
18425                 'reason': set([('fd', 'file'), ('file', 'f_mode')])},
18426                {'call': 'pwritev', 'reason': set([('fd', 'file')])},
18427                {'call': 'fchdir', 'reason': set([('fd', 'file')])},
18428                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
18429                {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
18430                {'call': 'accept4',
18431                 'reason': set([('fd', 'file'), ('file', 'f_mode')])},
18432                {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
18433                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
18434                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
18435                {'call': 'utimensat', 'reason': set([('fd', 'file')])},
18436                {'call': 'inotify_add_watch',
18437                 'reason': set([('fd', 'file')])},
18438                {'call': 'preadv2', 'reason': set([('fd', 'file')])},
18439                {'call': 'splice', 'reason': set([('fd', 'file')])},
18440                {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
18441                {'call': 'preadv', 'reason': set([('fd', 'file')])},
18442                {'call': 'getpeername', 'reason': set([('fd', 'file')])},
18443                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
18444                {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
18445                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
18446                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
18447                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
18448                {'call': 'ioctl', 'reason': set([('fd', 'file')])},
18449                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
18450                {'call': 'perf_event_open',
18451                 'reason': set([('fd', 'file'), ('file', 'f_mode')])},
18452                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
18453                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
18454                {'call': 'futimesat', 'reason': set([('fd', 'file')])},
18455                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
18456                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
18457                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
18458                {'call': 'open', 'reason': set([('file', 'f_mode')])},
18459                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
18460                {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
18461                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
18462                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
18463                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
18464                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
18465                {'call': 'listen', 'reason': set([('fd', 'file')])},
18466                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
18467                {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
18468                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
18469                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
18470                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
18471                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
18472                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
18473                {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
```

```
18474                {'call': 'llseek', 'reason': set([('fd', 'file')])},
18475                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
18476                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
18477                {'call': 'readv', 'reason': set([('fd', 'file')])},
18478                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
18479                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
18480                {'call': 'write', 'reason': set([('fd', 'file')])},
18481                {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
18482                {'call': 'sendfile', 'reason': set([('fd', 'file')])},
18483                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
18484                {'call': 'open_by_handle_at',
18485                 'reason': set([('file', 'f_mode')])},
18486                {'call': 'bind', 'reason': set([('fd', 'file')])},
18487                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
18488                {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
18489  'sendmmsg': [{'call': 'recvfrom', 'reason': set([('socket', 'file')])},
18490               {'call': 'sendto', 'reason': set([('socket', 'file')])},
18491               {'call': 'connect', 'reason': set([('socket', 'file')])},
18492               {'call': 'getsockname', 'reason': set([('socket', 'file')])},
18493               {'call': 'accept4', 'reason': set([('socket', 'file')])},
18494               {'call': 'getpeername', 'reason': set([('socket', 'file')])},
18495               {'call': 'setsockopt', 'reason': set([('socket', 'file')])},
18496               {'call': 'sendmsg', 'reason': set([('socket', 'file')])},
18497               {'call': 'shutdown', 'reason': set([('socket', 'file')])},
18498               {'call': 'getsockopt', 'reason': set([('socket', 'file')])},
18499               {'call': 'listen', 'reason': set([('socket', 'file')])},
18500               {'call': 'recvmsg', 'reason': set([('socket', 'file')])},
18501               {'call': 'sendmmsg', 'reason': set([('socket', 'file')])},
18502               {'call': 'bind', 'reason': set([('socket', 'file')])},
18503               {'call': 'recvmmsg', 'reason': set([('socket', 'file')])}],
18504  'sendmsg': [{'call': 'recvfrom', 'reason': set([('socket', 'file')])},
18505              {'call': 'sendto', 'reason': set([('socket', 'file')])},
18506              {'call': 'connect', 'reason': set([('socket', 'file')])},
18507              {'call': 'getsockname', 'reason': set([('socket', 'file')])},
18508              {'call': 'accept4', 'reason': set([('socket', 'file')])},
18509              {'call': 'getpeername', 'reason': set([('socket', 'file')])},
18510              {'call': 'setsockopt', 'reason': set([('socket', 'file')])},
18511              {'call': 'sendmsg', 'reason': set([('socket', 'file')])},
18512              {'call': 'shutdown', 'reason': set([('socket', 'file')])},
18513              {'call': 'getsockopt', 'reason': set([('socket', 'file')])},
18514              {'call': 'listen', 'reason': set([('socket', 'file')])},
18515              {'call': 'recvmsg', 'reason': set([('socket', 'file')])},
18516              {'call': 'sendmmsg', 'reason': set([('socket', 'file')])},
18517              {'call': 'bind', 'reason': set([('socket', 'file')])},
18518              {'call': 'recvmmsg', 'reason': set([('socket', 'file')])}],
18519  'sendto': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
18520             {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
18521             {'call': 'eventfd2', 'reason': set([('file', 'f_flags')])},
18522             {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
18523             {'call': 'swapoff', 'reason': set([('file', 'f_flags')])},
18524             {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
18525             {'call': 'readahead', 'reason': set([('fd', 'file')])},
18526             {'call': 'getdents', 'reason': set([('fd', 'file')])},
18527             {'call': 'writev', 'reason': set([('fd', 'file')])},
18528             {'call': 'preadv64', 'reason': set([('fd', 'file')])},
18529             {'call': 'fchmod', 'reason': set([('fd', 'file')])},
18530             {'call': 'pread64', 'reason': set([('fd', 'file')])},
18531             {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
18532             {'call': 'memfd_create', 'reason': set([('file', 'f_flags')])},
18533             {'call': 'remap_file_pages',
18534              'reason': set([('file', 'f_flags')])},
18535             {'call': 'dup3', 'reason': set([('file', 'f_flags')])},
18536             {'call': 'read', 'reason': set([('fd', 'file')])},
18537             {'call': 'fchown', 'reason': set([('fd', 'file')])},
18538             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
18539             {'call': 'utime', 'reason': set([('fd', 'file')])},
```



```
18540             {'call': 'fsync', 'reason': set([('fd', 'file')])},
18541             {'call': 'bpf', 'reason': set([('fd', 'file')])},
18542             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
18543             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
18544             {'call': 'sendto', 'reason': set([('fd', 'file')])},
18545             {'call': 'epoll_create1', 'reason': set([('file', 'f_flags')])},
18546             {'call': 'tee', 'reason': set([('fd', 'file')])},
18547             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
18548             {'call': 'lseek', 'reason': set([('fd', 'file')])},
18549             {'call': 'connect', 'reason': set([('fd', 'file')])},
18550             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
18551             {'call': 'epoll_ctl',
18552              'reason': set([('fd', 'file'), ('file', 'f_flags')])},
18553             {'call': 'flock',
18554              'reason': set([('fd', 'file'), ('file', 'f_flags')])},
18555             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
18556             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
18557             {'call': 'openat', 'reason': set([('file', 'f_flags')])},
18558             {'call': 'uselib', 'reason': set([('file', 'f_flags')])},
18559             {'call': 'accept4',
18560              'reason': set([('fd', 'file'), ('file', 'f_flags')])},
18561             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
18562             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
18563             {'call': 'socketpair', 'reason': set([('file', 'f_flags')])},
18564             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
18565             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
18566             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
18567             {'call': 'splice', 'reason': set([('fd', 'file')])},
18568             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
18569             {'call': 'preadv', 'reason': set([('fd', 'file')])},
18570             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
18571             {'call': 'shmat', 'reason': set([('file', 'f_flags')])},
18572             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
18573             {'call': 'socket', 'reason': set([('file', 'f_flags')])},
18574             {'call': 'pipe2', 'reason': set([('file', 'f_flags')])},
18575             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
18576             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
18577             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
18578             {'call': 'perf_event_open',
18579              'reason': set([('fd', 'file'), ('file', 'f_flags')])},
18580             {'call': 'shmdt', 'reason': set([('file', 'f_flags')])},
18581             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
18582             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
18583             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
18584             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
18585             {'call': 'acct', 'reason': set([('file', 'f_flags')])},
18586             {'call': 'open', 'reason': set([('file', 'f_flags')])},
18587             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
18588             {'call': 'mq_getsetattr',
18589              'reason': set([('fd', 'file'), ('file', 'f_flags')])},
18590             {'call': 'dup', 'reason': set([('file', 'f_flags')])},
18591             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
18592             {'call': 'setns', 'reason': set([('file', 'f_flags')])},
18593             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
18594             {'call': 'listen', 'reason': set([('fd', 'file')])},
18595             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
18596             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
18597             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
18598             {'call': 'shmctl', 'reason': set([('file', 'f_flags')])},
18599             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
18600             {'call': 'swapon', 'reason': set([('file', 'f_flags')])},
18601             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
18602             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
18603             {'call': 'llseek', 'reason': set([('fd', 'file')])},
18604             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_flags')])},
18605             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
```

```
18606             {'call': 'readv', 'reason': set([('fd', 'file')])},
18607             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
18608             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
18609             {'call': 'write', 'reason': set([('fd', 'file')])},
18610             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
18611             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
18612             {'call': 'mq_open', 'reason': set([('file', 'f_flags')])},
18613             {'call': 'open_by_handle_at',
18614              'reason': set([('file', 'f_flags')])},
18615             {'call': 'bind', 'reason': set([('fd', 'file')])},
18616             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
18617             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
18618  'set_mempolicy': [{'call': 'set_mempolicy',
18619                     'reason': set([('mempolicy', 'mode')])},
18620                    {'call': 'get_mempolicy',
18621                     'reason': set([('mempolicy', 'mode')])},
18622                    {'call': 'mbind', 'reason': set([('mempolicy', 'mode')])}],
18623  'set_thread_area': [{'call': 'keyctl',
18624                       'reason': set([('thread_struct', 'fsindex'),
18625                                      ('thread_struct', 'gsindex')])},
18626                      {'call': 'rt_sigtimedwait',
18627                       'reason': set([('thread_struct', 'fsindex'),
18628                                      ('thread_struct', 'gsindex')])},
18629                      {'call': 'msgrcv',
18630                       'reason': set([('thread_struct', 'fsindex'),
18631                                      ('thread_struct', 'gsindex')])},
18632                      {'call': 'kill',
18633                       'reason': set([('thread_struct', 'fsindex'),
18634                                      ('thread_struct', 'gsindex')])},
18635                      {'call': 'sched_getaffinity',
18636                       'reason': set([('thread_struct', 'fsindex'),
18637                                      ('thread_struct', 'gsindex')])},
18638                      {'call': 'arch_prctl',
18639                       'reason': set([('thread_struct', 'fsindex'),
18640                                      ('thread_struct', 'gsindex')])},
18641                      {'call': 'sched_setparam',
18642                       'reason': set([('thread_struct', 'fsindex'),
18643                                      ('thread_struct', 'gsindex')])},
18644                      {'call': 'ioprio_set',
18645                       'reason': set([('thread_struct', 'fsindex'),
18646                                      ('thread_struct', 'gsindex')])},
18647                      {'call': 'getppid',
18648                       'reason': set([('thread_struct', 'fsindex'),
18649                                      ('thread_struct', 'gsindex')])},
18650                      {'call': 'ioperm',
18651                       'reason': set([('thread_struct', 'fsindex'),
18652                                      ('thread_struct', 'gsindex')])},
18653                      {'call': 'mq_timedreceive',
18654                       'reason': set([('thread_struct', 'fsindex'),
18655                                      ('thread_struct', 'gsindex')])},
18656                      {'call': 'capget',
18657                       'reason': set([('thread_struct', 'fsindex'),
18658                                      ('thread_struct', 'gsindex')])},
18659                      {'call': 'sched_setaffinity',
18660                       'reason': set([('thread_struct', 'fsindex'),
18661                                      ('thread_struct', 'gsindex')])},
18662                      {'call': 'signal',
18663                       'reason': set([('thread_struct', 'fsindex'),
18664                                      ('thread_struct', 'gsindex')])},
18665                      {'call': 'semtimedop',
18666                       'reason': set([('thread_struct', 'fsindex'),
18667                                      ('thread_struct', 'gsindex')])},
18668                      {'call': 'umount',
18669                       'reason': set([('thread_struct', 'fsindex'),
18670                                      ('thread_struct', 'gsindex')])},
18671                      {'call': 'sched_rr_get_interval',
```



```
18672                       'reason': set([('thread_struct', 'fsindex'),
18673                                      ('thread_struct', 'gsindex')])},
18674                      {'call': 'rt_sigprocmask',
18675                       'reason': set([('thread_struct', 'fsindex'),
18676                                      ('thread_struct', 'gsindex')])},
18677                      {'call': 'setsid',
18678                       'reason': set([('thread_struct', 'fsindex'),
18679                                      ('thread_struct', 'gsindex')])},
18680                      {'call': 'sigaltstack',
18681                       'reason': set([('thread_struct', 'fsindex'),
18682                                      ('thread_struct', 'gsindex')])},
18683                      {'call': 'sched_setattr',
18684                       'reason': set([('thread_struct', 'fsindex'),
18685                                      ('thread_struct', 'gsindex')])},
18686                      {'call': 'migrate_pages',
18687                       'reason': set([('thread_struct', 'fsindex'),
18688                                      ('thread_struct', 'gsindex')])},
18689                      {'call': 'getitimer',
18690                       'reason': set([('thread_struct', 'fsindex'),
18691                                      ('thread_struct', 'gsindex')])},
18692                      {'call': 'setpgid',
18693                       'reason': set([('thread_struct', 'fsindex'),
18694                                      ('thread_struct', 'gsindex')])},
18695                      {'call': 'getsid',
18696                       'reason': set([('thread_struct', 'fsindex'),
18697                                      ('thread_struct', 'gsindex')])},
18698                      {'call': 'prlimit64',
18699                       'reason': set([('thread_struct', 'fsindex'),
18700                                      ('thread_struct', 'gsindex')])},
18701                      {'call': 'perf_event_open',
18702                       'reason': set([('thread_struct', 'fsindex'),
18703                                      ('thread_struct', 'gsindex')])},
18704                      {'call': 'rt_sigaction',
18705                       'reason': set([('thread_struct', 'fsindex'),
18706                                      ('thread_struct', 'gsindex')])},
18707                      {'call': 'getpgid',
18708                       'reason': set([('thread_struct', 'fsindex'),
18709                                      ('thread_struct', 'gsindex')])},
18710                      {'call': 'getpriority',
18711                       'reason': set([('thread_struct', 'fsindex'),
18712                                      ('thread_struct', 'gsindex')])},
18713                      {'call': 'sigaction',
18714                       'reason': set([('thread_struct', 'fsindex'),
18715                                      ('thread_struct', 'gsindex')])},
18716                      {'call': 'setns',
18717                       'reason': set([('thread_struct', 'fsindex'),
18718                                      ('thread_struct', 'gsindex')])},
18719                      {'call': 'fork',
18720                       'reason': set([('thread_struct', 'fsindex'),
18721                                      ('thread_struct', 'gsindex')])},
18722                      {'call': 'get_robust_list',
18723                       'reason': set([('thread_struct', 'fsindex'),
18724                                      ('thread_struct', 'gsindex')])},
18725                      {'call': 'mq_timedsend',
18726                       'reason': set([('thread_struct', 'fsindex'),
18727                                      ('thread_struct', 'gsindex')])},
18728                      {'call': 'sched_getscheduler',
18729                       'reason': set([('thread_struct', 'fsindex'),
18730                                      ('thread_struct', 'gsindex')])},
18731                      {'call': 'ptrace',
18732                       'reason': set([('thread_struct', 'fsindex'),
18733                                      ('thread_struct', 'gsindex')])},
18734                      {'call': 'sched_getattr',
18735                       'reason': set([('thread_struct', 'fsindex'),
18736                                      ('thread_struct', 'gsindex')])},
18737                      {'call': 'getrusage',
```

```
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'setitimer',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'ioprio_get',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'vfork',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'prctl',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'move_pages',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'setpriority',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'clone',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])},
                    {'call': 'sched_getparam',
                     'reason': set([('thread_struct', 'fsindex'),
                                    ('thread_struct', 'gsindex')])}],
 'set_trip_temp': [{'call': 'set_trip_temp',
                    'reason': set([('pkg_device', 'cpu'),
                                   ('pkg_device', 'tj_max')])},
                   {'call': 'get_trip_temp',
                    'reason': set([('pkg_device', 'cpu'),
                                   ('pkg_device', 'tj_max')])},
                   {'call': 'get_curr_temp',
                    'reason': set([('pkg_device', 'cpu'),
                                   ('pkg_device', 'tj_max')])}],
 'setdomainname': [{'call': 'setns',
                    'reason': set([('uts_namespace', 'user_ns')])}],
 'setfsgid': [{'call': 'keyctl',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'sched_getaffinity',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setparam',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setgid',
               'reason': set([('cred', 'egid'),
                              ('cred', 'fsgid'),
                              ('cred', 'gid'),
                              ('cred', 'sgid'),
                              ('cred', 'user_ns')])},
              {'call': 'ioprio_set',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'mq_timedreceive',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
```



```
              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setaffinity',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'setfsgid',
               'reason': set([('cred', 'fsgid'), ('cred', 'user_ns')])},
              {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'semtimedop',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sigaltstack',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setattr',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'migrate_pages',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setresgid',
               'reason': set([('cred', 'egid'),
                              ('cred', 'fsgid'),
                              ('cred', 'gid'),
                              ('cred', 'sgid'),
                              ('cred', 'user_ns')])},
              {'call': 'setregid',
               'reason': set([('cred', 'egid'),
                              ('cred', 'fsgid'),
                              ('cred', 'gid'),
                              ('cred', 'sgid'),
                              ('cred', 'user_ns')])},
              {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'prlimit64',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'perf_event_open',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
              {'call': 'rt_sigaction',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getpriority',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
              {'call': 'faccessat',
               'reason': set([('cred', 'fsgid'), ('cred', 'user_ns')])},
              {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
              {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
              {'call': 'get_robust_list',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'mq_timedsend',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_getscheduler',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_getattr',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setscheduler',
               'reason': set([('task_struct', 'cred')])},
```

```
18870               {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
18871               {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
18872               {'call': 'ioprio_get',
18873                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18874               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
18875               {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
18876               {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
18877               {'call': 'move_pages',
18878                'reason': set([('task_struct', 'cred')])},
18879               {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
18880               {'call': 'setpriority',
18881                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18882               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
18883               {'call': 'sched_getparam',
18884                'reason': set([('task_struct', 'cred')])}],
18885  'setfsuid': [{'call': 'keyctl',
18886                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18887               {'call': 'rt_sigtimedwait',
18888                'reason': set([('task_struct', 'cred')])},
18889               {'call': 'setfsuid',
18890                'reason': set([('cred', 'fsuid'), ('cred', 'user_ns')])},
18891               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
18892               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
18893               {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
18894               {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
18895               {'call': 'sched_getaffinity',
18896                'reason': set([('task_struct', 'cred')])},
18897               {'call': 'sched_setparam',
18898                'reason': set([('task_struct', 'cred')])},
18899               {'call': 'setgid', 'reason': set([('cred', 'user_ns')])},
18900               {'call': 'ioprio_set',
18901                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18902               {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
18903               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
18904               {'call': 'mq_timedreceive',
18905                'reason': set([('task_struct', 'cred')])},
18906               {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
18907               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
18908               {'call': 'sched_setaffinity',
18909                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18910               {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
18911               {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
18912               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
18913               {'call': 'setreuid',
18914                'reason': set([('cred', 'euid'),
18915                               ('cred', 'fsuid'),
18916                               ('cred', 'suid'),
18917                               ('cred', 'uid'),
18918                               ('cred', 'user_ns')])},
18919               {'call': 'semtimedop',
18920                'reason': set([('task_struct', 'cred')])},
18921               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
18922               {'call': 'sched_rr_get_interval',
18923                'reason': set([('task_struct', 'cred')])},
18924               {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
18925               {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
18926               {'call': 'rt_sigprocmask',
18927                'reason': set([('task_struct', 'cred')])},
18928               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
18929               {'call': 'sigaltstack',
18930                'reason': set([('task_struct', 'cred')])},
18931               {'call': 'sched_setattr',
18932                'reason': set([('task_struct', 'cred')])},
18933               {'call': 'migrate_pages',
18934                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18935               {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
```



```
18936               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
18937               {'call': 'setresgid', 'reason': set([('cred', 'user_ns')])},
18938               {'call': 'setregid', 'reason': set([('cred', 'user_ns')])},
18939               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
18940               {'call': 'prlimit64',
18941                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18942               {'call': 'perf_event_open',
18943                'reason': set([('task_struct', 'cred')])},
18944               {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
18945               {'call': 'rt_sigaction',
18946                'reason': set([('task_struct', 'cred')])},
18947               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
18948               {'call': 'getpriority',
18949                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18950               {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
18951               {'call': 'faccessat',
18952                'reason': set([('cred', 'fsuid'), ('cred', 'user_ns')])},
18953               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
18954               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
18955               {'call': 'get_robust_list',
18956                'reason': set([('task_struct', 'cred')])},
18957               {'call': 'mq_timedsend',
18958                'reason': set([('task_struct', 'cred')])},
18959               {'call': 'sched_getscheduler',
18960                'reason': set([('task_struct', 'cred')])},
18961               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
18962               {'call': 'sched_getattr',
18963                'reason': set([('task_struct', 'cred')])},
18964               {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
18965               {'call': 'sched_setscheduler',
18966                'reason': set([('task_struct', 'cred')])},
18967               {'call': 'setresuid',
18968                'reason': set([('cred', 'euid'),
18969                               ('cred', 'fsuid'),
18970                               ('cred', 'suid'),
18971                               ('cred', 'uid'),
18972                               ('cred', 'user_ns')])},
18973               {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
18974               {'call': 'ioprio_get',
18975                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18976               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
18977               {'call': 'setuid',
18978                'reason': set([('cred', 'euid'),
18979                               ('cred', 'fsuid'),
18980                               ('cred', 'suid'),
18981                               ('cred', 'uid'),
18982                               ('cred', 'user_ns')])},
18983               {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
18984               {'call': 'move_pages',
18985                'reason': set([('task_struct', 'cred')])},
18986               {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
18987               {'call': 'setpriority',
18988                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18989               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
18990               {'call': 'sched_getparam',
18991                'reason': set([('task_struct', 'cred')])}],
18992  'setgid': [{'call': 'keyctl',
18993              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
18994             {'call': 'rt_sigtimedwait',
18995              'reason': set([('task_struct', 'cred')])},
18996             {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
18997             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
18998             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
18999             {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
19000             {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
19001             {'call': 'sched_getaffinity',
```

```
19002              'reason': set([('task_struct', 'cred')])},
19003             {'call': 'sched_setparam',
19004              'reason': set([('task_struct', 'cred')])},
19005             {'call': 'setgid',
19006              'reason': set([('cred', 'gid'),
19007                             ('cred', 'sgid'),
19008                             ('cred', 'user_ns')])},
19009             {'call': 'ioprio_set',
19010              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19011             {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
19012             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
19013             {'call': 'mq_timedreceive',
19014              'reason': set([('task_struct', 'cred')])},
19015             {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
19016             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
19017             {'call': 'sched_setaffinity',
19018              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19019             {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
19020             {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
19021             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
19022             {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
19023             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
19024             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
19025             {'call': 'sched_rr_get_interval',
19026              'reason': set([('task_struct', 'cred')])},
19027             {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
19028             {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
19029             {'call': 'rt_sigprocmask',
19030              'reason': set([('task_struct', 'cred')])},
19031             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
19032             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
19033             {'call': 'sched_setattr',
19034              'reason': set([('task_struct', 'cred')])},
19035             {'call': 'migrate_pages',
19036              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19037             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
19038             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
19039             {'call': 'setresgid',
19040              'reason': set([('cred', 'gid'),
19041                             ('cred', 'sgid'),
19042                             ('cred', 'user_ns')])},
19043             {'call': 'setregid',
19044              'reason': set([('cred', 'gid'),
19045                             ('cred', 'sgid'),
19046                             ('cred', 'user_ns')])},
19047             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
19048             {'call': 'prlimit64',
19049              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19050             {'call': 'perf_event_open',
19051              'reason': set([('task_struct', 'cred')])},
19052             {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
19053             {'call': 'rt_sigaction',
19054              'reason': set([('task_struct', 'cred')])},
19055             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
19056             {'call': 'getpriority',
19057              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19058             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
19059             {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
19060             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
19061             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
19062             {'call': 'get_robust_list',
19063              'reason': set([('task_struct', 'cred')])},
19064             {'call': 'mq_timedsend',
19065              'reason': set([('task_struct', 'cred')])},
19066             {'call': 'sched_getscheduler',
19067              'reason': set([('task_struct', 'cred')])},
```



```
19068             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
19069             {'call': 'sched_getattr',
19070              'reason': set([('task_struct', 'cred')])},
19071             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
19072             {'call': 'sched_setscheduler',
19073              'reason': set([('task_struct', 'cred')])},
19074             {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
19075             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
19076             {'call': 'ioprio_get',
19077              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19078             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
19079             {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
19080             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
19081             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
19082             {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
19083             {'call': 'setpriority',
19084              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19085             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
19086             {'call': 'sched_getparam',
19087              'reason': set([('task_struct', 'cred')])}],
19088  'setgroups16': [{'call': 'setgroups16',
19089                   'reason': set([('group_info', 'ngroups')])},
19090                  {'call': 'setgroups',
19091                   'reason': set([('group_info', 'ngroups')])}],
19092  'sethostname': [{'call': 'setns',
19093                   'reason': set([('uts_namespace', 'user_ns')])}],
19094  'setitimer': [{'call': 'waitid',
19095                 'reason': set([('timeval', 'tv_sec'),
19096                                ('timeval', 'tv_usec')])},
19097                {'call': 'settimeofday',
19098                 'reason': set([('timeval', 'tv_sec'),
19099                                ('timeval', 'tv_usec')])},
19100                {'call': 'timer_create',
19101                 'reason': set([('signal_struct', 'it_real_incr')])},
19102                {'call': 'adjtimex',
19103                 'reason': set([('timeval', 'tv_sec'),
19104                                ('timeval', 'tv_usec')])},
19105                {'call': 'getitimer',
19106                 'reason': set([('itimerval', 'it_interval'),
19107                                ('itimerval', 'it_value'),
19108                                ('timeval', 'tv_sec'),
19109                                ('timeval', 'tv_usec')])},
19110                {'call': 'select',
19111                 'reason': set([('timeval', 'tv_sec'),
19112                                ('timeval', 'tv_usec')])},
19113                {'call': 'exit_group',
19114                 'reason': set([('signal_struct', 'it_real_incr')])},
19115                {'call': 'wait4',
19116                 'reason': set([('timeval', 'tv_sec'),
19117                                ('timeval', 'tv_usec')])},
19118                {'call': 'getrusage',
19119                 'reason': set([('timeval', 'tv_sec'),
19120                                ('timeval', 'tv_usec')])},
19121                {'call': 'setitimer',
19122                 'reason': set([('itimerval', 'it_interval'),
19123                                ('itimerval', 'it_value'),
19124                                ('signal_struct', 'it_real_incr'),
19125                                ('timeval', 'tv_sec'),
19126                                ('timeval', 'tv_usec')])},
19127                {'call': 'clock_adjtime',
19128                 'reason': set([('timeval', 'tv_sec'),
19129                                ('timeval', 'tv_usec')])},
19130                {'call': 'alarm',
19131                 'reason': set([('timeval', 'tv_sec'),
19132                                ('timeval', 'tv_usec')])},
19133                {'call': 'ppoll',
```

```
                'reason': set([('timeval', 'tv_sec'),
                               ('timeval', 'tv_usec')])}],
 'setns': [{'call': 'setns',
            'reason': set([('nsproxy', 'cgroup_ns'),
                           ('nsproxy', 'ipc_ns'),
                           ('nsproxy', 'mnt_ns'),
                           ('nsproxy', 'net_ns'),
                           ('nsproxy', 'pid_ns_for_children'),
                           ('nsproxy', 'uts_ns')])}],
 'setpgid': [{'call': 'keyctl',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'rt_sigtimedwait',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'msgrcv',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'kill',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sched_getaffinity',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sched_setparam',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'ioprio_set',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'getppid',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'mq_timedreceive',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'capget',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sched_setaffinity',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'signal',
              'reason': set([('task_struct', 'exit_signal'),
```



```
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
            {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'semtimedop',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'umount',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'timer_create',
             'reason': set([('signal_struct', 'leader')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'setsid',
             'reason': set([('signal_struct', 'leader'),
                            ('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'sigaltstack',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'getitimer',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'setpgid',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'getsid',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'signal')])},
            {'call': 'prlimit64',
             'reason': set([('task_struct', 'exit_signal'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'real_parent'),
```

```
                                ('task_struct', 'signal')])},
             {'call': 'perf_event_open',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'rt_sigaction',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'getpgid',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'exit_group',
              'reason': set([('signal_struct', 'leader')])},
             {'call': 'getpriority',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sigaction',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'setns',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'fork',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'get_robust_list',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'mq_timedsend',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sched_getscheduler',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'ptrace',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sched_getattr',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'getrusage',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
```



```
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sched_setscheduler',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
             {'call': 'setitimer',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'ioprio_get',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'vfork',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
             {'call': 'prctl',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'move_pages',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'setpriority',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'clone',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])},
             {'call': 'sched_getparam',
              'reason': set([('task_struct', 'exit_signal'),
                             ('task_struct', 'flags'),
                             ('task_struct', 'real_parent'),
                             ('task_struct', 'signal')])}],
 'setpriority': [{'call': 'keyctl',
                  'reason': set([('task_struct', 'cred'),
                                 ('task_struct', 'real_cred')])},
                 {'call': 'rt_sigtimedwait',
                  'reason': set([('task_struct', 'cred'),
                                 ('task_struct', 'real_cred')])},
                 {'call': 'msgrcv',
                  'reason': set([('task_struct', 'cred'),
                                 ('task_struct', 'real_cred')])},
                 {'call': 'kill',
                  'reason': set([('task_struct', 'cred'),
                                 ('task_struct', 'real_cred')])},
                 {'call': 'sched_getaffinity',
                  'reason': set([('task_struct', 'cred'),
                                 ('task_struct', 'real_cred')])},
                 {'call': 'sched_setparam',
                  'reason': set([('task_struct', 'cred'),
```

```
                                ('task_struct', 'real_cred')])},
                {'call': 'ioprio_set',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'capget',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'signal',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setreuid', 'reason': set([('cred', 'uid')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'umount',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpriority',
```



```
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setresuid', 'reason': set([('cred', 'uid')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setuid', 'reason': set([('cred', 'uid')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])}],
 'setregid': [{'call': 'keyctl',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
```

```
19530                {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
19531                {'call': 'sched_getaffinity',
19532                 'reason': set([('task_struct', 'cred')])},
19533                {'call': 'sched_setparam',
19534                 'reason': set([('task_struct', 'cred')])},
19535                {'call': 'setgid',
19536                 'reason': set([('cred', 'egid'),
19537                                ('cred', 'gid'),
19538                                ('cred', 'sgid'),
19539                                ('cred', 'user_ns')])},
19540                {'call': 'ioprio_set',
19541                 'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19542                {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
19543                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
19544                {'call': 'mq_timedreceive',
19545                 'reason': set([('task_struct', 'cred')])},
19546                {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
19547                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
19548                {'call': 'sched_setaffinity',
19549                 'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19550                {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
19551                {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
19552                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
19553                {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
19554                {'call': 'semtimedop',
19555                 'reason': set([('task_struct', 'cred')])},
19556                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
19557                {'call': 'sched_rr_get_interval',
19558                 'reason': set([('task_struct', 'cred')])},
19559                {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
19560                {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
19561                {'call': 'rt_sigprocmask',
19562                 'reason': set([('task_struct', 'cred')])},
19563                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
19564                {'call': 'sigaltstack',
19565                 'reason': set([('task_struct', 'cred')])},
19566                {'call': 'sched_setattr',
19567                 'reason': set([('task_struct', 'cred')])},
19568                {'call': 'migrate_pages',
19569                 'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19570                {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
19571                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
19572                {'call': 'setresgid',
19573                 'reason': set([('cred', 'egid'),
19574                                ('cred', 'gid'),
19575                                ('cred', 'sgid'),
19576                                ('cred', 'user_ns')])},
19577                {'call': 'setregid',
19578                 'reason': set([('cred', 'egid'),
19579                                ('cred', 'gid'),
19580                                ('cred', 'sgid'),
19581                                ('cred', 'user_ns')])},
19582                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
19583                {'call': 'prlimit64',
19584                 'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19585                {'call': 'perf_event_open',
19586                 'reason': set([('task_struct', 'cred')])},
19587                {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
19588                {'call': 'rt_sigaction',
19589                 'reason': set([('task_struct', 'cred')])},
19590                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
19591                {'call': 'getpriority',
19592                 'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19593                {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
19594                {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
19595                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
```



```
19596                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
19597                {'call': 'get_robust_list',
19598                 'reason': set([('task_struct', 'cred')])},
19599                {'call': 'mq_timedsend',
19600                 'reason': set([('task_struct', 'cred')])},
19601                {'call': 'sched_getscheduler',
19602                 'reason': set([('task_struct', 'cred')])},
19603                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
19604                {'call': 'sched_getattr',
19605                 'reason': set([('task_struct', 'cred')])},
19606                {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
19607                {'call': 'sched_setscheduler',
19608                 'reason': set([('task_struct', 'cred')])},
19609                {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
19610                {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
19611                {'call': 'ioprio_get',
19612                 'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19613                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
19614                {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
19615                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
19616                {'call': 'move_pages',
19617                 'reason': set([('task_struct', 'cred')])},
19618                {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
19619                {'call': 'setpriority',
19620                 'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
19621                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
19622                {'call': 'sched_getparam',
19623                 'reason': set([('task_struct', 'cred')])}],
19624  'setresgid': [{'call': 'keyctl',
19625                 'reason': set([('cred', 'user_ns'),
19626                                ('task_struct', 'cred')])},
19627                {'call': 'rt_sigtimedwait',
19628                 'reason': set([('task_struct', 'cred')])},
19629                {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
19630                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
19631                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
19632                {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
19633                {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
19634                {'call': 'sched_getaffinity',
19635                 'reason': set([('task_struct', 'cred')])},
19636                {'call': 'sched_setparam',
19637                 'reason': set([('task_struct', 'cred')])},
19638                {'call': 'setgid',
19639                 'reason': set([('cred', 'egid'),
19640                                ('cred', 'gid'),
19641                                ('cred', 'sgid'),
19642                                ('cred', 'user_ns')])},
19643                {'call': 'ioprio_set',
19644                 'reason': set([('cred', 'user_ns'),
19645                                ('task_struct', 'cred')])},
19646                {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
19647                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
19648                {'call': 'mq_timedreceive',
19649                 'reason': set([('task_struct', 'cred')])},
19650                {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
19651                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
19652                {'call': 'sched_setaffinity',
19653                 'reason': set([('cred', 'user_ns'),
19654                                ('task_struct', 'cred')])},
19655                {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
19656                {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
19657                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
19658                {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
19659                {'call': 'semtimedop',
19660                 'reason': set([('task_struct', 'cred')])},
19661                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
```

```
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'epoll_create1',
                 'reason': set([('cred', 'user_ns')])},
                {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'migrate_pages',
                 'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setresgid',
                 'reason': set([('cred', 'egid'),
                                ('cred', 'gid'),
                                ('cred', 'sgid'),
                                ('cred', 'user_ns')])},
                {'call': 'setregid',
                 'reason': set([('cred', 'egid'),
                                ('cred', 'gid'),
                                ('cred', 'sgid'),
                                ('cred', 'user_ns')])},
                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'prlimit64',
                 'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpriority',
                 'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
```



```
                {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
                {'call': 'setpriority',
                 'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred')])}],
 'setresuid': [{'call': 'keyctl',
                 'reason': set([('cred', 'user'),
                                ('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'rt_sigtimedwait',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setfsuid',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
                {'call': 'getresuid16',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'getresgid',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'sched_getaffinity',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setparam',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setgid',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'ioprio_set',
                 'reason': set([('cred', 'user'),
                                ('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'capset',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getresgid16',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('cred', 'user'),
                                ('cred', 'user_ns'),
                                ('task_struct', 'cred')])},
                {'call': 'setfsgid',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'unshare',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
                {'call': 'setreuid',
                 'reason': set([('cred', 'euid'),
                                ('cred', 'suid'),
                                ('cred', 'uid'),
                                ('cred', 'user'),
                                ('cred', 'user_ns')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'epoll_create1',
                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
                {'call': 'getresuid',
```

```
                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
               {'call': 'rt_sigprocmask',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaltstack',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'migrate_pages',
                'reason': set([('cred', 'user'),
                               ('cred', 'user_ns'),
                               ('task_struct', 'cred')])},
               {'call': 'getitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'setresgid',
                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
               {'call': 'setregid',
                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'prlimit64',
                'reason': set([('cred', 'user'),
                               ('cred', 'user_ns'),
                               ('task_struct', 'cred')])},
               {'call': 'perf_event_open',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getgroups16',
                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
               {'call': 'rt_sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'getpriority',
                'reason': set([('cred', 'user'),
                               ('cred', 'user_ns'),
                               ('task_struct', 'cred')])},
               {'call': 'sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'faccessat',
                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
               {'call': 'get_robust_list',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedsend',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getscheduler',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getrusage',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setscheduler',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setresuid',
                'reason': set([('cred', 'euid'),
                               ('cred', 'suid'),
                               ('cred', 'uid'),
                               ('cred', 'user'),
                               ('cred', 'user_ns')])},
               {'call': 'setitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ioprio_get',
                'reason': set([('cred', 'user'),
                               ('cred', 'user_ns'),
                               ('task_struct', 'cred')])},
```



```
               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
               {'call': 'setuid',
                'reason': set([('cred', 'euid'),
                               ('cred', 'suid'),
                               ('cred', 'uid'),
                               ('cred', 'user'),
                               ('cred', 'user_ns')])},
               {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
               {'call': 'move_pages',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getgroups',
                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
               {'call': 'setpriority',
                'reason': set([('cred', 'user'),
                               ('cred', 'user_ns'),
                               ('task_struct', 'cred')])},
               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getparam',
                'reason': set([('task_struct', 'cred')])}],
 'setreuid': [{'call': 'keyctl',
               'reason': set([('cred', 'user'),
                              ('cred', 'user_ns'),
                              ('task_struct', 'cred')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setfsuid',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getresuid16',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'getresgid',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'sched_getaffinity',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setparam',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setgid',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'ioprio_set',
               'reason': set([('cred', 'user'),
                              ('cred', 'user_ns'),
                              ('task_struct', 'cred')])},
              {'call': 'capset',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'mq_timedreceive',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getresgid16',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setaffinity',
               'reason': set([('cred', 'user'),
                              ('cred', 'user_ns'),
                              ('task_struct', 'cred')])},
              {'call': 'setfsgid',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'unshare',
               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setreuid',
               'reason': set([('cred', 'euid'),
                              ('cred', 'suid'),
                              ('cred', 'uid'),
                              ('cred', 'user'),
                              ('cred', 'user_ns')])},
```

```
19926                {'call': 'semtimedop',
19927                 'reason': set([('task_struct', 'cred')])},
19928                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
19929                {'call': 'sched_rr_get_interval',
19930                 'reason': set([('task_struct', 'cred')])},
19931                {'call': 'epoll_create1',
19932                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
19933                {'call': 'getresuid',
19934                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
19935                {'call': 'rt_sigprocmask',
19936                 'reason': set([('task_struct', 'cred')])},
19937                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
19938                {'call': 'sigaltstack',
19939                 'reason': set([('task_struct', 'cred')])},
19940                {'call': 'sched_setattr',
19941                 'reason': set([('task_struct', 'cred')])},
19942                {'call': 'migrate_pages',
19943                 'reason': set([('cred', 'user'),
19944                                ('cred', 'user_ns'),
19945                                ('task_struct', 'cred')])},
19946                {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
19947                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
19948                {'call': 'setresgid',
19949                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
19950                {'call': 'setregid',
19951                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
19952                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
19953                {'call': 'prlimit64',
19954                 'reason': set([('cred', 'user'),
19955                                ('cred', 'user_ns'),
19956                                ('task_struct', 'cred')])},
19957                {'call': 'perf_event_open',
19958                 'reason': set([('task_struct', 'cred')])},
19959                {'call': 'getgroups16',
19960                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
19961                {'call': 'rt_sigaction',
19962                 'reason': set([('task_struct', 'cred')])},
19963                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
19964                {'call': 'getpriority',
19965                 'reason': set([('cred', 'user'),
19966                                ('cred', 'user_ns'),
19967                                ('task_struct', 'cred')])},
19968                {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
19969                {'call': 'faccessat',
19970                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
19971                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
19972                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
19973                {'call': 'get_robust_list',
19974                 'reason': set([('task_struct', 'cred')])},
19975                {'call': 'mq_timedsend',
19976                 'reason': set([('task_struct', 'cred')])},
19977                {'call': 'sched_getscheduler',
19978                 'reason': set([('task_struct', 'cred')])},
19979                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
19980                {'call': 'sched_getattr',
19981                 'reason': set([('task_struct', 'cred')])},
19982                {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
19983                {'call': 'sched_setscheduler',
19984                 'reason': set([('task_struct', 'cred')])},
19985                {'call': 'setresuid',
19986                 'reason': set([('cred', 'euid'),
19987                                ('cred', 'suid'),
19988                                ('cred', 'uid'),
19989                                ('cred', 'user'),
19990                                ('cred', 'user_ns')])},
19991                {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
```



```
19992                {'call': 'ioprio_get',
19993                 'reason': set([('cred', 'user'),
19994                                ('cred', 'user_ns'),
19995                                ('task_struct', 'cred')])},
19996                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
19997                {'call': 'setuid',
19998                 'reason': set([('cred', 'euid'),
19999                                ('cred', 'suid'),
20000                                ('cred', 'uid'),
20001                                ('cred', 'user'),
20002                                ('cred', 'user_ns')])},
20003                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
20004                {'call': 'move_pages',
20005                 'reason': set([('task_struct', 'cred')])},
20006                {'call': 'getgroups',
20007                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20008                {'call': 'setpriority',
20009                 'reason': set([('cred', 'user'),
20010                                ('cred', 'user_ns'),
20011                                ('task_struct', 'cred')])},
20012                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
20013                {'call': 'sched_getparam',
20014                 'reason': set([('task_struct', 'cred')])}],
20015  'setrlimit': [{'call': 'keyctl',
20016                 'reason': set([('task_struct', 'group_leader'),
20017                                ('task_struct', 'sighand')])},
20018                 {'call': 'rt_sigtimedwait',
20019                  'reason': set([('task_struct', 'group_leader'),
20020                                 ('task_struct', 'sighand')])},
20021                 {'call': 'msgrcv',
20022                  'reason': set([('task_struct', 'group_leader'),
20023                                 ('task_struct', 'sighand')])},
20024                 {'call': 'kill',
20025                  'reason': set([('task_struct', 'group_leader'),
20026                                 ('task_struct', 'sighand')])},
20027                 {'call': 'sched_getaffinity',
20028                  'reason': set([('task_struct', 'group_leader'),
20029                                 ('task_struct', 'sighand')])},
20030                 {'call': 'sched_setparam',
20031                  'reason': set([('task_struct', 'group_leader'),
20032                                 ('task_struct', 'sighand')])},
20033                 {'call': 'ioprio_set',
20034                  'reason': set([('task_struct', 'group_leader'),
20035                                 ('task_struct', 'sighand')])},
20036                 {'call': 'getppid',
20037                  'reason': set([('task_struct', 'group_leader'),
20038                                 ('task_struct', 'sighand')])},
20039                 {'call': 'mq_timedreceive',
20040                  'reason': set([('task_struct', 'group_leader'),
20041                                 ('task_struct', 'sighand')])},
20042                 {'call': 'capget',
20043                  'reason': set([('task_struct', 'group_leader'),
20044                                 ('task_struct', 'sighand')])},
20045                 {'call': 'sched_setaffinity',
20046                  'reason': set([('task_struct', 'group_leader'),
20047                                 ('task_struct', 'sighand')])},
20048                 {'call': 'signal',
20049                  'reason': set([('task_struct', 'group_leader'),
20050                                 ('task_struct', 'sighand')])},
20051                 {'call': 'semtimedop',
20052                  'reason': set([('task_struct', 'group_leader'),
20053                                 ('task_struct', 'sighand')])},
20054                 {'call': 'umount',
20055                  'reason': set([('task_struct', 'group_leader'),
20056                                 ('task_struct', 'sighand')])},
20057                 {'call': 'sched_rr_get_interval',
```

```
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setrlimit',
                 'reason': set([('compat_rlimit', 'rlim_cur'),
                                ('compat_rlimit', 'rlim_max'),
                                ('rlimit', 'rlim_cur'),
                                ('rlimit', 'rlim_max')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'old_getrlimit',
                 'reason': set([('rlimit', 'rlim_cur'),
                                ('rlimit', 'rlim_max')])},
                {'call': 'prlimit64',
                 'reason': set([('rlimit', 'rlim_cur'),
                                ('rlimit', 'rlim_max'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
```



```
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getrlimit',
                 'reason': set([('compat_rlimit', 'rlim_cur'),
                                ('compat_rlimit', 'rlim_max')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])}],
 'setsid': [{'call': 'timer_create',
             'reason': set([('signal_struct', 'leader')])},
            {'call': 'setsid', 'reason': set([('signal_struct', 'leader')])},
            {'call': 'exit_group',
             'reason': set([('signal_struct', 'leader')])}],
 'setsockopt': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
                {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
                {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
                {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
                {'call': 'readahead', 'reason': set([('fd', 'file')])},
                {'call': 'getdents', 'reason': set([('fd', 'file')])},
                {'call': 'writev', 'reason': set([('fd', 'file')])},
                {'call': 'preadv64', 'reason': set([('fd', 'file')])},
                {'call': 'fchmod', 'reason': set([('fd', 'file')])},
                {'call': 'pread64', 'reason': set([('fd', 'file')])},
                {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
                {'call': 'read', 'reason': set([('fd', 'file')])},
                {'call': 'fchown', 'reason': set([('fd', 'file')])},
                {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
                {'call': 'utime', 'reason': set([('fd', 'file')])},
                {'call': 'fsync', 'reason': set([('fd', 'file')])},
                {'call': 'bpf', 'reason': set([('fd', 'file')])},
                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
```

```
20190                 {'call': 'sendto', 'reason': set([('fd', 'file')])},
20191                 {'call': 'tee', 'reason': set([('fd', 'file')])},
20192                 {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
20193                 {'call': 'lseek', 'reason': set([('fd', 'file')])},
20194                 {'call': 'connect', 'reason': set([('fd', 'file')])},
20195                 {'call': 'getsockname', 'reason': set([('fd', 'file')])},
20196                 {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
20197                 {'call': 'flock', 'reason': set([('fd', 'file')])},
20198                 {'call': 'pwritev', 'reason': set([('fd', 'file')])},
20199                 {'call': 'fchdir', 'reason': set([('fd', 'file')])},
20200                 {'call': 'accept4',
20201                  'reason': set([('fd', 'file'),
20202                                 ('proto_ops', 'compat_setsockopt')])},
20203                 {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
20204                 {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
20205                 {'call': 'utimensat', 'reason': set([('fd', 'file')])},
20206                 {'call': 'inotify_add_watch',
20207                  'reason': set([('fd', 'file')])},
20208                 {'call': 'preadv2', 'reason': set([('fd', 'file')])},
20209                 {'call': 'splice', 'reason': set([('fd', 'file')])},
20210                 {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
20211                 {'call': 'preadv', 'reason': set([('fd', 'file')])},
20212                 {'call': 'getpeername', 'reason': set([('fd', 'file')])},
20213                 {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
20214                 {'call': 'fcntl', 'reason': set([('fd', 'file')])},
20215                 {'call': 'ioctl', 'reason': set([('fd', 'file')])},
20216                 {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
20217                 {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
20218                 {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
20219                 {'call': 'futimesat', 'reason': set([('fd', 'file')])},
20220                 {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
20221                 {'call': 'shutdown', 'reason': set([('fd', 'file')])},
20222                 {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
20223                 {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
20224                 {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
20225                 {'call': 'getdents64', 'reason': set([('fd', 'file')])},
20226                 {'call': 'listen', 'reason': set([('fd', 'file')])},
20227                 {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
20228                 {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
20229                 {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
20230                 {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
20231                 {'call': 'fallocate', 'reason': set([('fd', 'file')])},
20232                 {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
20233                 {'call': 'llseek', 'reason': set([('fd', 'file')])},
20234                 {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
20235                 {'call': 'readv', 'reason': set([('fd', 'file')])},
20236                 {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
20237                 {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
20238                 {'call': 'write', 'reason': set([('fd', 'file')])},
20239                 {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
20240                 {'call': 'sendfile', 'reason': set([('fd', 'file')])},
20241                 {'call': 'bind', 'reason': set([('fd', 'file')])},
20242                 {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
20243                 {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
20244  'settimeofday': [{'call': 'waitid',
20245                    'reason': set([('timeval', 'tv_sec'),
20246                                   ('timeval', 'tv_usec')])},
20247                   {'call': 'settimeofday',
20248                    'reason': set([('timeval', 'tv_sec'),
20249                                   ('timeval', 'tv_usec'),
20250                                   ('timezone', 'tz_minuteswest')])},
20251                   {'call': 'adjtimex',
20252                    'reason': set([('timeval', 'tv_sec'),
20253                                   ('timeval', 'tv_usec')])},
20254                   {'call': 'getitimer',
20255                    'reason': set([('timeval', 'tv_sec'),
```



```
20256                                   ('timeval', 'tv_usec')])},
20257                   {'call': 'select',
20258                    'reason': set([('timeval', 'tv_sec'),
20259                                   ('timeval', 'tv_usec')])},
20260                   {'call': 'wait4',
20261                    'reason': set([('timeval', 'tv_sec'),
20262                                   ('timeval', 'tv_usec')])},
20263                   {'call': 'getrusage',
20264                    'reason': set([('timeval', 'tv_sec'),
20265                                   ('timeval', 'tv_usec')])},
20266                   {'call': 'setitimer',
20267                    'reason': set([('timeval', 'tv_sec'),
20268                                   ('timeval', 'tv_usec')])},
20269                   {'call': 'clock_adjtime',
20270                    'reason': set([('timeval', 'tv_sec'),
20271                                   ('timeval', 'tv_usec')])},
20272                   {'call': 'alarm',
20273                    'reason': set([('timeval', 'tv_sec'),
20274                                   ('timeval', 'tv_usec')])},
20275                   {'call': 'ppoll',
20276                    'reason': set([('timeval', 'tv_sec'),
20277                                   ('timeval', 'tv_usec')])}],
20278  'setuid': [{'call': 'keyctl',
20279              'reason': set([('cred', 'user'),
20280                             ('cred', 'user_ns'),
20281                             ('task_struct', 'cred')])},
20282             {'call': 'rt_sigtimedwait',
20283              'reason': set([('task_struct', 'cred')])},
20284             {'call': 'setfsuid',
20285              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20286             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
20287             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
20288             {'call': 'getresuid16',
20289              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20290             {'call': 'getresgid',
20291              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20292             {'call': 'sched_getaffinity',
20293              'reason': set([('task_struct', 'cred')])},
20294             {'call': 'sched_setparam',
20295              'reason': set([('task_struct', 'cred')])},
20296             {'call': 'setgid',
20297              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20298             {'call': 'ioprio_set',
20299              'reason': set([('cred', 'user'),
20300                             ('cred', 'user_ns'),
20301                             ('task_struct', 'cred')])},
20302             {'call': 'capset',
20303              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20304             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
20305             {'call': 'mq_timedreceive',
20306              'reason': set([('task_struct', 'cred')])},
20307             {'call': 'getresgid16',
20308              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20309             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
20310             {'call': 'sched_setaffinity',
20311              'reason': set([('cred', 'user'),
20312                             ('cred', 'user_ns'),
20313                             ('task_struct', 'cred')])},
20314             {'call': 'setfsgid',
20315              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20316             {'call': 'unshare',
20317              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
20318             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
20319             {'call': 'setreuid',
20320              'reason': set([('cred', 'suid'),
20321                             ('cred', 'uid'),
```

```
                              ('cred', 'user'),
                              ('cred', 'user_ns')])},
            {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
            {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'epoll_create1',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'getresuid',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'migrate_pages',
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setresgid',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'setregid',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'prlimit64',
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'perf_event_open',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getgroups16',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getpriority',
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
            {'call': 'faccessat',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
            {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setresuid',
             'reason': set([('cred', 'suid'),
                            ('cred', 'uid'),
                            ('cred', 'user'),
                            ('cred', 'user_ns')])},
            {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
            {'call': 'ioprio_get',
```



```
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setuid',
             'reason': set([('cred', 'suid'),
                            ('cred', 'uid'),
                            ('cred', 'user'),
                            ('cred', 'user_ns')])},
            {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
            {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getgroups',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'setpriority',
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_getparam',
             'reason': set([('task_struct', 'cred')])}],
 'setxattr': [{'call': 'eventfd2',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'swapoff',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'pivot_root',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'memfd_create',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'remap_file_pages',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'dup3',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'unshare',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'epoll_create1',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'epoll_ctl',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'flock',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'openat',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'lookup_dcookie',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'uselib',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'accept4',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'socketpair',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'getcwd',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'shmat',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'socket',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'pipe2',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'perf_event_open',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'shmdt',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'quotactl',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'acct',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```

```
20454               {'call': 'open',
20455                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
20456               {'call': 'dup',
20457                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
20458               {'call': 'setns',
20459                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
20460               {'call': 'shmctl',
20461                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
20462               {'call': 'swapon',
20463                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
20464               {'call': 'mmap_pgoff',
20465                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
20466               {'call': 'mq_open',
20467                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
20468               {'call': 'open_by_handle_at',
20469                'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
20470  'shmat': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
20471            {'call': 'rt_sigtimedwait',
20472             'reason': set([('task_struct', 'mm')])},
20473            {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
20474            {'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
20475            {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
20476            {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
20477            {'call': 'sched_getaffinity',
20478             'reason': set([('task_struct', 'mm')])},
20479            {'call': 'sched_setparam', 'reason': set([('task_struct', 'mm')])},
20480            {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
20481            {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
20482            {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
20483            {'call': 'remap_file_pages', 'reason': set([('path', 'dentry')])},
20484            {'call': 'dup3', 'reason': set([('path', 'dentry')])},
20485            {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
20486            {'call': 'mq_timedreceive',
20487             'reason': set([('task_struct', 'mm')])},
20488            {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
20489            {'call': 'sched_setaffinity',
20490             'reason': set([('task_struct', 'mm')])},
20491            {'call': 'unshare', 'reason': set([('path', 'dentry')])},
20492            {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
20493            {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
20494            {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
20495            {'call': 'sched_rr_get_interval',
20496             'reason': set([('task_struct', 'mm')])},
20497            {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
20498            {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
20499            {'call': 'flock', 'reason': set([('path', 'dentry')])},
20500            {'call': 'openat', 'reason': set([('path', 'dentry')])},
20501            {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
20502            {'call': 'uselib', 'reason': set([('path', 'dentry')])},
20503            {'call': 'rt_sigprocmask', 'reason': set([('task_struct', 'mm')])},
20504            {'call': 'accept4', 'reason': set([('path', 'dentry')])},
20505            {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
20506            {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
20507            {'call': 'sched_setattr', 'reason': set([('task_struct', 'mm')])},
20508            {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
20509            {'call': 'migrate_pages', 'reason': set([('task_struct', 'mm')])},
20510            {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
20511            {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
20512            {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
20513            {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
20514            {'call': 'shmat',
20515             'reason': set([('path', 'dentry'),
20516                            ('shmid_kernel', 'shm_file')])},
20517            {'call': 'socket', 'reason': set([('path', 'dentry')])},
20518            {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
20519            {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
```



```
20520            {'call': 'perf_event_open',
20521             'reason': set([('path', 'dentry'), ('task_struct', 'mm')])},
20522            {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
20523            {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
20524            {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
20525            {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
20526            {'call': 'acct', 'reason': set([('path', 'dentry')])},
20527            {'call': 'open', 'reason': set([('path', 'dentry')])},
20528            {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
20529            {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
20530            {'call': 'dup', 'reason': set([('path', 'dentry')])},
20531            {'call': 'setns',
20532             'reason': set([('path', 'dentry'), ('task_struct', 'mm')])},
20533            {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
20534            {'call': 'get_robust_list',
20535             'reason': set([('task_struct', 'mm')])},
20536            {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
20537            {'call': 'sched_getscheduler',
20538             'reason': set([('task_struct', 'mm')])},
20539            {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
20540            {'call': 'shmctl',
20541             'reason': set([('path', 'dentry'),
20542                            ('shmid_kernel', 'shm_file')])},
20543            {'call': 'swapon', 'reason': set([('path', 'dentry')])},
20544            {'call': 'sched_getattr', 'reason': set([('task_struct', 'mm')])},
20545            {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
20546            {'call': 'sched_setscheduler',
20547             'reason': set([('task_struct', 'mm')])},
20548            {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
20549            {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
20550            {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
20551            {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
20552            {'call': 'prctl', 'reason': set([('task_struct', 'mm')])},
20553            {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
20554            {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
20555            {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
20556            {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
20557            {'call': 'sched_getparam', 'reason': set([('task_struct', 'mm')])},
20558            {'call': 'open_by_handle_at', 'reason': set([('path', 'dentry')])}],
20559  'shmctl': [{'call': 'keyctl',
20560              'reason': set([('mm_segment_t', 'seg'),
20561                             ('task_struct', 'cred')])},
20562             {'call': 'rt_sigtimedwait',
20563              'reason': set([('mm_segment_t', 'seg'),
20564                             ('task_struct', 'cred')])},
20565             {'call': 'msgrcv',
20566              'reason': set([('ipc_namespace', 'user_ns'),
20567                             ('kern_ipc_perm', 'deleted'),
20568                             ('kern_ipc_perm', 'mode'),
20569                             ('mm_segment_t', 'seg'),
20570                             ('task_struct', 'cred')])},
20571             {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
20572             {'call': 'mq_unlink',
20573              'reason': set([('ipc_namespace', 'user_ns')])},
20574             {'call': 'kill',
20575              'reason': set([('mm_segment_t', 'seg'),
20576                             ('task_struct', 'cred')])},
20577             {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
20578             {'call': 'msgget', 'reason': set([('ipc_namespace', 'user_ns')])},
20579             {'call': 'sched_getaffinity',
20580              'reason': set([('mm_segment_t', 'seg'),
20581                             ('task_struct', 'cred')])},
20582             {'call': 'sched_setparam',
20583              'reason': set([('mm_segment_t', 'seg'),
20584                             ('task_struct', 'cred')])},
20585             {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
```

```
20586             {'call': 'ioprio_set',
20587              'reason': set([('mm_segment_t', 'seg'),
20588                             ('task_struct', 'cred')])},
20589             {'call': 'remap_file_pages', 'reason': set([('file', 'f_op')])},
20590             {'call': 'dup3', 'reason': set([('file', 'f_op')])},
20591             {'call': 'getppid',
20592              'reason': set([('mm_segment_t', 'seg'),
20593                             ('task_struct', 'cred')])},
20594             {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
20595             {'call': 'mq_timedreceive',
20596              'reason': set([('mm_segment_t', 'seg'),
20597                             ('task_struct', 'cred')])},
20598             {'call': 'capget',
20599              'reason': set([('mm_segment_t', 'seg'),
20600                             ('task_struct', 'cred')])},
20601             {'call': 'sched_setaffinity',
20602              'reason': set([('mm_segment_t', 'seg'),
20603                             ('task_struct', 'cred')])},
20604             {'call': 'signal',
20605              'reason': set([('mm_segment_t', 'seg'),
20606                             ('task_struct', 'cred')])},
20607             {'call': 'semtimedop',
20608              'reason': set([('ipc_namespace', 'user_ns'),
20609                             ('kern_ipc_perm', 'deleted'),
20610                             ('kern_ipc_perm', 'mode'),
20611                             ('mm_segment_t', 'seg'),
20612                             ('task_struct', 'cred')])},
20613             {'call': 'umount',
20614              'reason': set([('mm_segment_t', 'seg'),
20615                             ('task_struct', 'cred')])},
20616             {'call': 'sched_rr_get_interval',
20617              'reason': set([('mm_segment_t', 'seg'),
20618                             ('task_struct', 'cred')])},
20619             {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
20620             {'call': 'semctl',
20621              'reason': set([('ipc_namespace', 'user_ns'),
20622                             ('kern_ipc_perm', 'deleted'),
20623                             ('kern_ipc_perm', 'mode')])},
20624             {'call': 'epoll_ctl', 'reason': set([('file', 'f_op')])},
20625             {'call': 'flock', 'reason': set([('file', 'f_op')])},
20626             {'call': 'openat', 'reason': set([('file', 'f_op')])},
20627             {'call': 'shmget', 'reason': set([('ipc_namespace', 'user_ns')])},
20628             {'call': 'uselib', 'reason': set([('file', 'f_op')])},
20629             {'call': 'rt_sigprocmask',
20630              'reason': set([('mm_segment_t', 'seg'),
20631                             ('task_struct', 'cred')])},
20632             {'call': 'accept4', 'reason': set([('file', 'f_op')])},
20633             {'call': 'msgctl',
20634              'reason': set([('ipc_namespace', 'user_ns'),
20635                             ('kern_ipc_perm', 'deleted'),
20636                             ('kern_ipc_perm', 'mode')])},
20637             {'call': 'setsid',
20638              'reason': set([('mm_segment_t', 'seg'),
20639                             ('task_struct', 'cred')])},
20640             {'call': 'sigaltstack',
20641              'reason': set([('mm_segment_t', 'seg'),
20642                             ('task_struct', 'cred')])},
20643             {'call': 'sched_setattr',
20644              'reason': set([('mm_segment_t', 'seg'),
20645                             ('task_struct', 'cred')])},
20646             {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
20647             {'call': 'migrate_pages',
20648              'reason': set([('mm_segment_t', 'seg'),
20649                             ('task_struct', 'cred')])},
20650             {'call': 'getitimer',
20651              'reason': set([('mm_segment_t', 'seg'),
```



```
20652                             ('task_struct', 'cred')])},
20653             {'call': 'setpgid',
20654              'reason': set([('mm_segment_t', 'seg'),
20655                             ('task_struct', 'cred')])},
20656             {'call': 'semget', 'reason': set([('ipc_namespace', 'user_ns')])},
20657             {'call': 'getsid',
20658              'reason': set([('mm_segment_t', 'seg'),
20659                             ('task_struct', 'cred')])},
20660             {'call': 'shmat',
20661              'reason': set([('file', 'f_op'),
20662                             ('ipc_namespace', 'user_ns'),
20663                             ('kern_ipc_perm', 'deleted'),
20664                             ('kern_ipc_perm', 'mode')])},
20665             {'call': 'socket', 'reason': set([('file', 'f_op')])},
20666             {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
20667             {'call': 'prlimit64',
20668              'reason': set([('mm_segment_t', 'seg'),
20669                             ('task_struct', 'cred')])},
20670             {'call': 'perf_event_open',
20671              'reason': set([('file', 'f_op'),
20672                             ('mm_segment_t', 'seg'),
20673                             ('task_struct', 'cred')])},
20674             {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
20675             {'call': 'rt_sigaction',
20676              'reason': set([('mm_segment_t', 'seg'),
20677                             ('task_struct', 'cred')])},
20678             {'call': 'getpgid',
20679              'reason': set([('mm_segment_t', 'seg'),
20680                             ('task_struct', 'cred')])},
20681             {'call': 'acct', 'reason': set([('file', 'f_op')])},
20682             {'call': 'open', 'reason': set([('file', 'f_op')])},
20683             {'call': 'getpriority',
20684              'reason': set([('mm_segment_t', 'seg'),
20685                             ('task_struct', 'cred')])},
20686             {'call': 'sigaction',
20687              'reason': set([('mm_segment_t', 'seg'),
20688                             ('task_struct', 'cred')])},
20689             {'call': 'dup', 'reason': set([('file', 'f_op')])},
20690             {'call': 'setns',
20691              'reason': set([('file', 'f_op'),
20692                             ('ipc_namespace', 'user_ns'),
20693                             ('mm_segment_t', 'seg'),
20694                             ('task_struct', 'cred')])},
20695             {'call': 'fork',
20696              'reason': set([('mm_segment_t', 'seg'),
20697                             ('task_struct', 'cred')])},
20698             {'call': 'get_robust_list',
20699              'reason': set([('mm_segment_t', 'seg'),
20700                             ('task_struct', 'cred')])},
20701             {'call': 'mq_timedsend',
20702              'reason': set([('mm_segment_t', 'seg'),
20703                             ('task_struct', 'cred')])},
20704             {'call': 'sched_getscheduler',
20705              'reason': set([('mm_segment_t', 'seg'),
20706                             ('task_struct', 'cred')])},
20707             {'call': 'ptrace',
20708              'reason': set([('mm_segment_t', 'seg'),
20709                             ('task_struct', 'cred')])},
20710             {'call': 'shmctl',
20711              'reason': set([('file', 'f_op'),
20712                             ('ipc_namespace', 'user_ns'),
20713                             ('kern_ipc_perm', 'deleted'),
20714                             ('kern_ipc_perm', 'mode'),
20715                             ('shminfo64', 'shmmax')])},
20716             {'call': 'swapon', 'reason': set([('file', 'f_op')])},
20717             {'call': 'sched_getattr',
```

```
20718              'reason': set([('mm_segment_t', 'seg'),
20719                             ('task_struct', 'cred')])},
20720             {'call': 'getrusage',
20721              'reason': set([('mm_segment_t', 'seg'),
20722                             ('task_struct', 'cred')])},
20723             {'call': 'sched_setscheduler',
20724              'reason': set([('mm_segment_t', 'seg'),
20725                             ('task_struct', 'cred')])},
20726             {'call': 'setitimer',
20727              'reason': set([('mm_segment_t', 'seg'),
20728                             ('task_struct', 'cred')])},
20729             {'call': 'ioprio_get',
20730              'reason': set([('mm_segment_t', 'seg'),
20731                             ('task_struct', 'cred')])},
20732             {'call': 'vfork',
20733              'reason': set([('mm_segment_t', 'seg'),
20734                             ('task_struct', 'cred')])},
20735             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
20736             {'call': 'prctl',
20737              'reason': set([('mm_segment_t', 'seg'),
20738                             ('task_struct', 'cred')])},
20739             {'call': 'move_pages',
20740              'reason': set([('mm_segment_t', 'seg'),
20741                             ('task_struct', 'cred')])},
20742             {'call': 'msgsnd',
20743              'reason': set([('ipc_namespace', 'user_ns'),
20744                             ('kern_ipc_perm', 'deleted'),
20745                             ('kern_ipc_perm', 'mode')])},
20746             {'call': 'setpriority',
20747              'reason': set([('mm_segment_t', 'seg'),
20748                             ('task_struct', 'cred')])},
20749             {'call': 'clone',
20750              'reason': set([('mm_segment_t', 'seg'),
20751                             ('task_struct', 'cred')])},
20752             {'call': 'mq_open',
20753              'reason': set([('file', 'f_op'), ('ipc_namespace', 'user_ns')])},
20754             {'call': 'sched_getparam',
20755              'reason': set([('mm_segment_t', 'seg'),
20756                             ('task_struct', 'cred')])},
20757             {'call': 'open_by_handle_at', 'reason': set([('file', 'f_op')])}],
20758  'shmdt': [{'call': 'remap_file_pages',
20759             'reason': set([('vm_area_struct', 'vm_file'),
20760                            ('vm_area_struct', 'vm_ops'),
20761                            ('vm_area_struct', 'vm_pgoff')])},
20762            {'call': 'shmdt',
20763             'reason': set([('vm_area_struct', 'vm_file'),
20764                            ('vm_area_struct', 'vm_ops'),
20765                            ('vm_area_struct', 'vm_pgoff')])},
20766            {'call': 'brk',
20767             'reason': set([('vm_area_struct', 'vm_file'),
20768                            ('vm_area_struct', 'vm_ops'),
20769                            ('vm_area_struct', 'vm_pgoff')])},
20770            {'call': 'get_mempolicy',
20771             'reason': set([('vm_area_struct', 'vm_file'),
20772                            ('vm_area_struct', 'vm_ops'),
20773                            ('vm_area_struct', 'vm_pgoff')])},
20774            {'call': 'munlockall',
20775             'reason': set([('vm_area_struct', 'vm_file'),
20776                            ('vm_area_struct', 'vm_ops'),
20777                            ('vm_area_struct', 'vm_pgoff')])},
20778            {'call': 'pkey_mprotect',
20779             'reason': set([('vm_area_struct', 'vm_file'),
20780                            ('vm_area_struct', 'vm_ops'),
20781                            ('vm_area_struct', 'vm_pgoff')])},
20782            {'call': 'madvise',
20783             'reason': set([('vm_area_struct', 'vm_file'),
```



```
20784                            ('vm_area_struct', 'vm_ops'),
20785                            ('vm_area_struct', 'vm_pgoff')])},
20786            {'call': 'mprotect',
20787             'reason': set([('vm_area_struct', 'vm_file'),
20788                            ('vm_area_struct', 'vm_ops'),
20789                            ('vm_area_struct', 'vm_pgoff')])},
20790            {'call': 'mremap',
20791             'reason': set([('vm_area_struct', 'vm_file'),
20792                            ('vm_area_struct', 'vm_ops'),
20793                            ('vm_area_struct', 'vm_pgoff')])},
20794            {'call': 'prctl',
20795             'reason': set([('vm_area_struct', 'vm_file'),
20796                            ('vm_area_struct', 'vm_ops'),
20797                            ('vm_area_struct', 'vm_pgoff')])},
20798            {'call': 'munlock',
20799             'reason': set([('vm_area_struct', 'vm_file'),
20800                            ('vm_area_struct', 'vm_ops'),
20801                            ('vm_area_struct', 'vm_pgoff')])},
20802            {'call': 'mincore',
20803             'reason': set([('vm_area_struct', 'vm_file'),
20804                            ('vm_area_struct', 'vm_ops'),
20805                            ('vm_area_struct', 'vm_pgoff')])},
20806            {'call': 'mlockall',
20807             'reason': set([('vm_area_struct', 'vm_file'),
20808                            ('vm_area_struct', 'vm_ops'),
20809                            ('vm_area_struct', 'vm_pgoff')])}],
20810  'shutdown': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
20811               {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
20812               {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
20813               {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
20814               {'call': 'readahead', 'reason': set([('fd', 'file')])},
20815               {'call': 'getdents', 'reason': set([('fd', 'file')])},
20816               {'call': 'writev', 'reason': set([('fd', 'file')])},
20817               {'call': 'preadv64', 'reason': set([('fd', 'file')])},
20818               {'call': 'fchmod', 'reason': set([('fd', 'file')])},
20819               {'call': 'pread64', 'reason': set([('fd', 'file')])},
20820               {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
20821               {'call': 'read', 'reason': set([('fd', 'file')])},
20822               {'call': 'fchown', 'reason': set([('fd', 'file')])},
20823               {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
20824               {'call': 'utime', 'reason': set([('fd', 'file')])},
20825               {'call': 'fsync', 'reason': set([('fd', 'file')])},
20826               {'call': 'bpf', 'reason': set([('fd', 'file')])},
20827               {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
20828               {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
20829               {'call': 'sendto', 'reason': set([('fd', 'file')])},
20830               {'call': 'tee', 'reason': set([('fd', 'file')])},
20831               {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
20832               {'call': 'lseek', 'reason': set([('fd', 'file')])},
20833               {'call': 'connect', 'reason': set([('fd', 'file')])},
20834               {'call': 'getsockname', 'reason': set([('fd', 'file')])},
20835               {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
20836               {'call': 'flock', 'reason': set([('fd', 'file')])},
20837               {'call': 'pwritev', 'reason': set([('fd', 'file')])},
20838               {'call': 'fchdir', 'reason': set([('fd', 'file')])},
20839               {'call': 'accept4', 'reason': set([('fd', 'file')])},
20840               {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
20841               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
20842               {'call': 'utimensat', 'reason': set([('fd', 'file')])},
20843               {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
20844               {'call': 'preadv2', 'reason': set([('fd', 'file')])},
20845               {'call': 'splice', 'reason': set([('fd', 'file')])},
20846               {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
20847               {'call': 'preadv', 'reason': set([('fd', 'file')])},
20848               {'call': 'getpeername', 'reason': set([('fd', 'file')])},
20849               {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
```

```
20850               {'call': 'fcntl', 'reason': set([('fd', 'file')])},
20851               {'call': 'ioctl', 'reason': set([('fd', 'file')])},
20852               {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
20853               {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
20854               {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
20855               {'call': 'futimesat', 'reason': set([('fd', 'file')])},
20856               {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
20857               {'call': 'shutdown', 'reason': set([('fd', 'file')])},
20858               {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
20859               {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
20860               {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
20861               {'call': 'getdents64', 'reason': set([('fd', 'file')])},
20862               {'call': 'listen', 'reason': set([('fd', 'file')])},
20863               {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
20864               {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
20865               {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
20866               {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
20867               {'call': 'fallocate', 'reason': set([('fd', 'file')])},
20868               {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
20869               {'call': 'llseek', 'reason': set([('fd', 'file')])},
20870               {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
20871               {'call': 'readv', 'reason': set([('fd', 'file')])},
20872               {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
20873               {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
20874               {'call': 'write', 'reason': set([('fd', 'file')])},
20875               {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
20876               {'call': 'sendfile', 'reason': set([('fd', 'file')])},
20877               {'call': 'bind', 'reason': set([('fd', 'file')])},
20878               {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
20879               {'call': 'sendfile64', 'reason': set([('fd', 'file')])}],
20880  'sigaction': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
20881                {'call': 'rt_sigtimedwait',
20882                 'reason': set([('mm_segment_t', 'seg')])},
20883                {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
20884                {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
20885                {'call': 'sched_getaffinity',
20886                 'reason': set([('mm_segment_t', 'seg')])},
20887                {'call': 'sched_setparam',
20888                 'reason': set([('mm_segment_t', 'seg')])},
20889                {'call': 'ioprio_set',
20890                 'reason': set([('mm_segment_t', 'seg')])},
20891                {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
20892                {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
20893                {'call': 'mq_timedreceive',
20894                 'reason': set([('mm_segment_t', 'seg')])},
20895                {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
20896                {'call': 'sched_setaffinity',
20897                 'reason': set([('mm_segment_t', 'seg')])},
20898                {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
20899                {'call': 'semtimedop',
20900                 'reason': set([('mm_segment_t', 'seg')])},
20901                {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
20902                {'call': 'sched_rr_get_interval',
20903                 'reason': set([('mm_segment_t', 'seg')])},
20904                {'call': 'rt_sigprocmask',
20905                 'reason': set([('mm_segment_t', 'seg')])},
20906                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
20907                {'call': 'sigaltstack',
20908                 'reason': set([('mm_segment_t', 'seg')])},
20909                {'call': 'sched_setattr',
20910                 'reason': set([('mm_segment_t', 'seg')])},
20911                {'call': 'migrate_pages',
20912                 'reason': set([('mm_segment_t', 'seg')])},
20913                {'call': 'getitimer',
20914                 'reason': set([('mm_segment_t', 'seg')])},
20915                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
```



```
20916               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
20917               {'call': 'prlimit64',
20918                'reason': set([('mm_segment_t', 'seg')])},
20919               {'call': 'perf_event_open',
20920                'reason': set([('mm_segment_t', 'seg')])},
20921               {'call': 'rt_sigaction',
20922                'reason': set([('mm_segment_t', 'seg')])},
20923               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
20924               {'call': 'getpriority',
20925                'reason': set([('mm_segment_t', 'seg')])},
20926               {'call': 'sigaction',
20927                'reason': set([('mm_segment_t', 'seg')])},
20928               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
20929               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
20930               {'call': 'get_robust_list',
20931                'reason': set([('mm_segment_t', 'seg')])},
20932               {'call': 'mq_timedsend',
20933                'reason': set([('mm_segment_t', 'seg')])},
20934               {'call': 'sched_getscheduler',
20935                'reason': set([('mm_segment_t', 'seg')])},
20936               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
20937               {'call': 'sched_getattr',
20938                'reason': set([('mm_segment_t', 'seg')])},
20939               {'call': 'getrusage',
20940                'reason': set([('mm_segment_t', 'seg')])},
20941               {'call': 'sched_setscheduler',
20942                'reason': set([('mm_segment_t', 'seg')])},
20943               {'call': 'setitimer',
20944                'reason': set([('mm_segment_t', 'seg')])},
20945               {'call': 'ioprio_get',
20946                'reason': set([('mm_segment_t', 'seg')])},
20947               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
20948               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
20949               {'call': 'move_pages',
20950                'reason': set([('mm_segment_t', 'seg')])},
20951               {'call': 'setpriority',
20952                'reason': set([('mm_segment_t', 'seg')])},
20953               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
20954               {'call': 'sched_getparam',
20955                'reason': set([('mm_segment_t', 'seg')])}],
20956  'signalfd4': [{'call': 'syncfs',
20957                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20958                {'call': 'vmsplice',
20959                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20960                {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
20961                {'call': 'pwritev64',
20962                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20963                {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
20964                {'call': 'fremovexattr',
20965                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20966                {'call': 'readahead',
20967                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20968                {'call': 'getdents',
20969                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20970                {'call': 'writev',
20971                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20972                {'call': 'preadv64',
20973                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20974                {'call': 'fchmod',
20975                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20976                {'call': 'pread64',
20977                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20978                {'call': 'signalfd4',
20979                 'reason': set([('fd', 'file'), ('fd', 'flags')])},
20980                {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
20981                {'call': 'remap_file_pages',
```

```
20982                'reason': set([('file', 'f_op')])},
20983               {'call': 'dup3', 'reason': set([('file', 'f_op')])},
20984               {'call': 'read',
20985                'reason': set([('fd', 'file'), ('fd', 'flags')])},
20986               {'call': 'fchown',
20987                'reason': set([('fd', 'file'), ('fd', 'flags')])},
20988               {'call': 'mq_timedreceive',
20989                'reason': set([('fd', 'file'), ('fd', 'flags')])},
20990               {'call': 'utime',
20991                'reason': set([('fd', 'file'), ('fd', 'flags')])},
20992               {'call': 'fsync',
20993                'reason': set([('fd', 'file'), ('fd', 'flags')])},
20994               {'call': 'bpf',
20995                'reason': set([('fd', 'file'), ('fd', 'flags')])},
20996               {'call': 'recvfrom',
20997                'reason': set([('fd', 'file'), ('fd', 'flags')])},
20998               {'call': 'fsetxattr',
20999                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21000               {'call': 'sendto',
21001                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21002               {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
21003               {'call': 'tee',
21004                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21005               {'call': 'sync_file_range',
21006                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21007               {'call': 'lseek',
21008                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21009               {'call': 'connect',
21010                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21011               {'call': 'getsockname',
21012                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21013               {'call': 'epoll_ctl',
21014                'reason': set([('fd', 'file'),
21015                               ('fd', 'flags'),
21016                               ('file', 'f_op')])},
21017               {'call': 'flock',
21018                'reason': set([('fd', 'file'),
21019                               ('fd', 'flags'),
21020                               ('file', 'f_op')])},
21021               {'call': 'pwritev',
21022                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21023               {'call': 'fchdir',
21024                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21025               {'call': 'openat', 'reason': set([('file', 'f_op')])},
21026               {'call': 'uselib', 'reason': set([('file', 'f_op')])},
21027               {'call': 'accept4',
21028                'reason': set([('fd', 'file'),
21029                               ('fd', 'flags'),
21030                               ('file', 'f_op')])},
21031               {'call': 'old_readdir',
21032                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21033               {'call': 'inotify_rm_watch',
21034                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21035               {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
21036               {'call': 'utimensat',
21037                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21038               {'call': 'inotify_add_watch',
21039                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21040               {'call': 'preadv2',
21041                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21042               {'call': 'splice',
21043                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21044               {'call': 'ftruncate',
21045                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21046               {'call': 'preadv',
21047                'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
21048               {'call': 'getpeername',
21049                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21050               {'call': 'shmat', 'reason': set([('file', 'f_op')])},
21051               {'call': 'setsockopt',
21052                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21053               {'call': 'socket', 'reason': set([('file', 'f_op')])},
21054               {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
21055               {'call': 'fcntl',
21056                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21057               {'call': 'ioctl',
21058                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21059               {'call': 'pwrite64',
21060                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21061               {'call': 'perf_event_open',
21062                'reason': set([('fd', 'file'),
21063                               ('fd', 'flags'),
21064                               ('file', 'f_op')])},
21065               {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
21066               {'call': 'pwritev64v2',
21067                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21068               {'call': 'futimesat',
21069                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21070               {'call': 'pwritev2',
21071                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21072               {'call': 'shutdown',
21073                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21074               {'call': 'acct', 'reason': set([('file', 'f_op')])},
21075               {'call': 'open', 'reason': set([('file', 'f_op')])},
21076               {'call': 'getsockopt',
21077                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21078               {'call': 'mq_getsetattr',
21079                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21080               {'call': 'dup', 'reason': set([('file', 'f_op')])},
21081               {'call': 'fdatasync',
21082                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21083               {'call': 'setns', 'reason': set([('file', 'f_op')])},
21084               {'call': 'getdents64',
21085                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21086               {'call': 'listen',
21087                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21088               {'call': 'copy_file_range',
21089                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21090               {'call': 'mq_timedsend',
21091                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21092               {'call': 'fgetxattr',
21093                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21094               {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
21095               {'call': 'fcntl64',
21096                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21097               {'call': 'swapon', 'reason': set([('file', 'f_op')])},
21098               {'call': 'fallocate',
21099                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21100               {'call': 'epoll_wait',
21101                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21102               {'call': 'llseek',
21103                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21104               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
21105               {'call': 'preadv64v2',
21106                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21107               {'call': 'readv',
21108                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21109               {'call': 'fstatfs',
21110                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21111               {'call': 'fstatfs64',
21112                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21113               {'call': 'write',
```

```
21114                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21115               {'call': 'mq_notify',
21116                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21117               {'call': 'sendfile',
21118                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21119               {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
21120               {'call': 'open_by_handle_at',
21121                'reason': set([('file', 'f_op')])},
21122               {'call': 'bind',
21123                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21124               {'call': 'flistxattr',
21125                'reason': set([('fd', 'file'), ('fd', 'flags')])},
21126               {'call': 'sendfile64',
21127                'reason': set([('fd', 'file'), ('fd', 'flags')])}],
21128  'splice': [{'call': 'syncfs',
21129              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21130             {'call': 'vmsplice',
21131              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21132             {'call': 'eventfd2',
21133              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21134             {'call': 'pwritev64',
21135              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21136             {'call': 'swapoff',
21137              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21138             {'call': 'fremovexattr',
21139              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21140             {'call': 'readahead',
21141              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21142             {'call': 'getdents',
21143              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21144             {'call': 'writev',
21145              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21146             {'call': 'preadv64',
21147              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21148             {'call': 'fchmod',
21149              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21150             {'call': 'pread64',
21151              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21152             {'call': 'signalfd4',
21153              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21154             {'call': 'memfd_create',
21155              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21156             {'call': 'remap_file_pages',
21157              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21158             {'call': 'dup3',
21159              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21160             {'call': 'read',
21161              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21162             {'call': 'fchown',
21163              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21164             {'call': 'mq_timedreceive',
21165              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21166             {'call': 'utime',
21167              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21168             {'call': 'fsync',
21169              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21170             {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
21171             {'call': 'recvfrom',
21172              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21173             {'call': 'fsetxattr',
21174              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21175             {'call': 'sendto',
21176              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21177             {'call': 'epoll_create1',
21178              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21179             {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
21180             {'call': 'sync_file_range',
21181              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21182             {'call': 'lseek',
21183              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21184             {'call': 'connect',
21185              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21186             {'call': 'getsockname',
21187              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21188             {'call': 'epoll_ctl',
21189              'reason': set([('fd', 'file'),
21190                             ('fd', 'flags'),
21191                             ('file', 'f_flags'),
21192                             ('file', 'f_mode')])},
21193             {'call': 'flock',
21194              'reason': set([('fd', 'file'),
21195                             ('fd', 'flags'),
21196                             ('file', 'f_flags'),
21197                             ('file', 'f_mode')])},
21198             {'call': 'pwritev',
21199              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21200             {'call': 'fchdir',
21201              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21202             {'call': 'openat',
21203              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21204             {'call': 'uselib',
21205              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21206             {'call': 'accept4',
21207              'reason': set([('fd', 'file'),
21208                             ('fd', 'flags'),
21209                             ('file', 'f_flags'),
21210                             ('file', 'f_mode')])},
21211             {'call': 'old_readdir',
21212              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21213             {'call': 'inotify_rm_watch',
21214              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21215             {'call': 'socketpair',
21216              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21217             {'call': 'utimensat',
21218              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21219             {'call': 'inotify_add_watch',
21220              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21221             {'call': 'preadv2',
21222              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21223             {'call': 'splice',
21224              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21225             {'call': 'ftruncate',
21226              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21227             {'call': 'preadv',
21228              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21229             {'call': 'getpeername',
21230              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21231             {'call': 'shmat',
21232              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21233             {'call': 'setsockopt',
21234              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21235             {'call': 'socket',
21236              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21237             {'call': 'pipe2',
21238              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21239             {'call': 'fcntl',
21240              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21241             {'call': 'ioctl',
21242              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21243             {'call': 'pwrite64',
21244              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21245             {'call': 'perf_event_open',
```

```
21246              'reason': set([('fd', 'file'),
21247                             ('fd', 'flags'),
21248                             ('file', 'f_flags'),
21249                             ('file', 'f_mode')])},
21250             {'call': 'shmdt',
21251              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21252             {'call': 'pwritev64v2',
21253              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21254             {'call': 'futimesat',
21255              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21256             {'call': 'pwritev2',
21257              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21258             {'call': 'shutdown',
21259              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21260             {'call': 'acct',
21261              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21262             {'call': 'open',
21263              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21264             {'call': 'getsockopt',
21265              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21266             {'call': 'mq_getsetattr',
21267              'reason': set([('fd', 'file'),
21268                             ('fd', 'flags'),
21269                             ('file', 'f_flags')])},
21270             {'call': 'dup',
21271              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21272             {'call': 'fdatasync',
21273              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21274             {'call': 'setns',
21275              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21276             {'call': 'getdents64',
21277              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21278             {'call': 'listen',
21279              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21280             {'call': 'copy_file_range',
21281              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21282             {'call': 'mq_timedsend',
21283              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21284             {'call': 'fgetxattr',
21285              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21286             {'call': 'shmctl',
21287              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21288             {'call': 'fcntl64',
21289              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21290             {'call': 'swapon',
21291              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21292             {'call': 'fallocate',
21293              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21294             {'call': 'epoll_wait',
21295              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21296             {'call': 'llseek',
21297              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21298             {'call': 'mmap_pgoff',
21299              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21300             {'call': 'preadv64v2',
21301              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21302             {'call': 'readv',
21303              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21304             {'call': 'fstatfs',
21305              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21306             {'call': 'fstatfs64',
21307              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21308             {'call': 'write',
21309              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21310             {'call': 'mq_notify',
21311              'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
21312             {'call': 'sendfile',
21313              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21314             {'call': 'mq_open',
21315              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21316             {'call': 'open_by_handle_at',
21317              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
21318             {'call': 'bind',
21319              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21320             {'call': 'flistxattr',
21321              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21322             {'call': 'sendfile64',
21323              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
21324  'stat': [{'call': 'lstat',
21325            'reason': set([('__old_kernel_stat', 'st_ino'),
21326                           ('__old_kernel_stat', 'st_nlink')])},
21327           {'call': 'stat',
21328            'reason': set([('__old_kernel_stat', 'st_ino'),
21329                           ('__old_kernel_stat', 'st_nlink')])},
21330           {'call': 'fstat',
21331            'reason': set([('__old_kernel_stat', 'st_ino'),
21332                           ('__old_kernel_stat', 'st_nlink'),
21333                           ('kstat', 'dev'),
21334                           ('kstat', 'ino'),
21335                           ('kstat', 'nlink'),
21336                           ('kstat', 'rdev')])},
21337           {'call': 'newfstat',
21338            'reason': set([('kstat', 'dev'),
21339                           ('kstat', 'ino'),
21340                           ('kstat', 'nlink'),
21341                           ('kstat', 'rdev')])}],
21342  'statfs': [{'call': 'ustat',
21343              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
21344             {'call': 'fstatfs',
21345              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
21346             {'call': 'statfs',
21347              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
21348             {'call': 'fstatfs64',
21349              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
21350             {'call': 'statfs64',
21351              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])}],
21352  'statfs64': [{'call': 'ustat',
21353                'reason': set([('kstatfs', 'f_ffree'),
21354                               ('kstatfs', 'f_files')])},
21355               {'call': 'fstatfs',
21356                'reason': set([('kstatfs', 'f_ffree'),
21357                               ('kstatfs', 'f_files')])},
21358               {'call': 'statfs',
21359                'reason': set([('kstatfs', 'f_ffree'),
21360                               ('kstatfs', 'f_files')])},
21361               {'call': 'fstatfs64',
21362                'reason': set([('kstatfs', 'f_ffree'),
21363                               ('kstatfs', 'f_files')])},
21364               {'call': 'statfs64',
21365                'reason': set([('kstatfs', 'f_ffree'),
21366                               ('kstatfs', 'f_files')])}],
21367  'swapoff': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
21368              {'call': 'rt_sigtimedwait',
21369               'reason': set([('task_struct', 'mm')])},
21370              {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
21371              {'call': 'eventfd2', 'reason': set([('file', 'f_mapping')])},
21372              {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
21373              {'call': 'swapoff',
21374               'reason': set([('file', 'f_mapping'),
21375                              ('page', 'private'),
21376                              ('swap_info_struct', 'cluster_info'),
21377                              ('swap_info_struct', 'flags'),
```

```
                              (’swap_info_struct’, ’inuse_pages’),
                              (’swap_info_struct’, ’pages’),
                              (’swap_info_struct’, ’prio’),
                              (’swap_info_struct’, ’swap_map’),
                              (’swp_entry_t’, ’val’)])},
             {’call’: ’sched_getaffinity’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sched_setparam’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’memfd_create’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’ioprio_set’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’remap_file_pages’,
              ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’dup3’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’getppid’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’mq_timedreceive’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’capget’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sched_setaffinity’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’signal’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’semtimedop’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’umount’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sched_rr_get_interval’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’epoll_create1’,
              ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’epoll_ctl’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’flock’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’openat’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’uselib’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’rt_sigprocmask’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’accept4’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’setsid’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sigaltstack’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sched_setattr’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’socketpair’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’migrate_pages’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’getitimer’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’setpgid’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’getsid’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’shmat’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’socket’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’pipe2’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’prlimit64’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’perf_event_open’,
              ’reason’: set([(’file’, ’f_mapping’), (’task_struct’, ’mm’)])},
             {’call’: ’shmdt’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’rt_sigaction’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’getpgid’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’acct’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’open’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’getpriority’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sigaction’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’dup’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’setns’,
              ’reason’: set([(’file’, ’f_mapping’), (’task_struct’, ’mm’)])},
             {’call’: ’fork’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’get_robust_list’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’mq_timedsend’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sched_getscheduler’,
              ’reason’: set([(’task_struct’, ’mm’)])},
```



```
             {’call’: ’ptrace’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’shmctl’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’swapon’,
              ’reason’: set([(’file’, ’f_mapping’),
                             (’page’, ’private’),
                             (’swap_info_struct’, ’cluster_info’),
                             (’swap_info_struct’, ’flags’),
                             (’swap_info_struct’, ’inuse_pages’),
                             (’swap_info_struct’, ’pages’),
                             (’swap_info_struct’, ’prio’),
                             (’swap_info_struct’, ’swap_map’)])},
             {’call’: ’sched_getattr’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’getrusage’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’sched_setscheduler’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’setitimer’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’ioprio_get’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’vfork’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’mmap_pgoff’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’prctl’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’move_pages’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’setpriority’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’clone’, ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’mq_open’, ’reason’: set([(’file’, ’f_mapping’)])},
             {’call’: ’sched_getparam’,
              ’reason’: set([(’task_struct’, ’mm’)])},
             {’call’: ’open_by_handle_at’,
              ’reason’: set([(’file’, ’f_mapping’)])}],
 ’swapon’: [{’call’: ’mq_unlink’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’swapoff’,
             ’reason’: set([(’inode’, ’i_flags’),
                            (’swap_info_struct’, ’bdev’),
                            (’swap_info_struct’, ’flags’),
                            (’swap_info_struct’, ’percpu_cluster’),
                            (’swap_info_struct’, ’type’)])},
            {’call’: ’fchmod’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’memfd_create’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’readlinkat’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’fchown’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’mq_timedreceive’,
             ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’uselib’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’fchmodat’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’inotify_add_watch’,
             ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’ftruncate’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’ioctl’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’linkat’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’unlink’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’mq_getsetattr’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’faccessat’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’mq_timedsend’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’swapon’,
             ’reason’: set([(’inode’, ’i_flags’),
                            (’swap_info_struct’, ’bdev’),
                            (’swap_info_struct’, ’flags’),
                            (’swap_info_struct’, ’percpu_cluster’),
                            (’swap_info_struct’, ’type’)])},
            {’call’: ’fchownat’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’mq_notify’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’sendfile’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’unlinkat’, ’reason’: set([(’inode’, ’i_flags’)])},
            {’call’: ’sendfile64’, ’reason’: set([(’inode’, ’i_flags’)])}],
 ’symlinkat’: [{’call’: ’sysfs’,
                ’reason’: set([(’filename’, ’name’),
```

```
                                 ('filename', 'refcnt')])},
                {'call': 'mq_unlink',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'swapoff',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'openat',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'uselib',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'renameat2',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'symlinkat',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'quotactl',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'acct',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'open',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'unlink',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'rmdir',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'swapon',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'mq_open',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])},
                {'call': 'unlinkat',
                 'reason': set([('filename', 'name'),
                                ('filename', 'refcnt')])}],
 'sync_file_range': [{'call': 'syncfs',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'vmsplice',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'pwritev64',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'fremovexattr',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'readahead',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'getdents',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'writev',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'preadv64',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'fchmod',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'pread64',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'signalfd4',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'read',
```



```
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'fchown',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'mq_timedreceive',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'utime',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'fsync',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'bpf',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'recvfrom',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'fsetxattr',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'sendto',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'tee',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'sync_file_range',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'lseek',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'connect',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'getsockname',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'epoll_ctl',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'flock',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'pwritev',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'fchdir',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'accept4',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'old_readdir',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'inotify_rm_watch',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'utimensat',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'inotify_add_watch',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'preadv2',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'splice',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'ftruncate',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'preadv',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'getpeername',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'setsockopt',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'fcntl',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'ioctl',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'pwrite64',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'perf_event_open',
                      'reason': set([('fd', 'file'), ('fd', 'flags')])},
                     {'call': 'pwritev64v2',
```

```
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'futimesat',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'pwritev2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'shutdown',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'getsockopt',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_getsetattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fdatasync',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'getdents64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'listen',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'copy_file_range',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fgetxattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fcntl64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fallocate',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'epoll_wait',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'llseek',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'preadv64v2',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'readv',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fstatfs',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'fstatfs64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'write',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'mq_notify',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'sendfile',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'bind',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'flistxattr',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])},
                    {'call': 'sendfile64',
                     'reason': set([('fd', 'file'), ('fd', 'flags')])}],
 'syncfs': [{'call': 'syncfs',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'vmsplice',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'pwritev64',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'fremovexattr',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'readahead',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'getdents',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'writev',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'preadv64',
```



```
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'fchmod',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'pread64',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'signalfd4',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'read',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'fchown',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'mq_timedreceive',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'utime',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'fsync',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'recvfrom',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'fsetxattr',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'sendto',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'sync_file_range',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'lseek',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'connect',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'getsockname',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'epoll_ctl',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'flock',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'pwritev',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'fchdir',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'accept4',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'old_readdir',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'inotify_rm_watch',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'utimensat',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'inotify_add_watch',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'preadv2',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'splice',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'ftruncate',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'preadv',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'getpeername',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'setsockopt',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'fcntl',
             'reason': set([('fd', 'file'), ('fd', 'flags')])},
            {'call': 'ioctl',
```

```
21774              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21775             {'call': 'pwrite64',
21776              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21777             {'call': 'perf_event_open',
21778              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21779             {'call': 'pwritev64v2',
21780              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21781             {'call': 'futimesat',
21782              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21783             {'call': 'pwritev2',
21784              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21785             {'call': 'shutdown',
21786              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21787             {'call': 'getsockopt',
21788              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21789             {'call': 'mq_getsetattr',
21790              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21791             {'call': 'fdatasync',
21792              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21793             {'call': 'getdents64',
21794              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21795             {'call': 'listen',
21796              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21797             {'call': 'copy_file_range',
21798              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21799             {'call': 'mq_timedsend',
21800              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21801             {'call': 'fgetxattr',
21802              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21803             {'call': 'fcntl64',
21804              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21805             {'call': 'fallocate',
21806              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21807             {'call': 'epoll_wait',
21808              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21809             {'call': 'llseek',
21810              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21811             {'call': 'preadv64v2',
21812              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21813             {'call': 'readv',
21814              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21815             {'call': 'fstatfs',
21816              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21817             {'call': 'fstatfs64',
21818              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21819             {'call': 'write',
21820              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21821             {'call': 'mq_notify',
21822              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21823             {'call': 'sendfile',
21824              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21825             {'call': 'bind',
21826              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21827             {'call': 'flistxattr',
21828              'reason': set([('fd', 'file'), ('fd', 'flags')])},
21829             {'call': 'sendfile64',
21830              'reason': set([('fd', 'file'), ('fd', 'flags')])}],
21831  'sysctl': [{'call': 'sysctl',
21832              'reason': set([('__sysctl_args', 'oldlenp'),
21833                             ('__sysctl_args', 'oldval'),
21834                             ('compat_sysctl_args', 'oldlenp'),
21835                             ('compat_sysctl_args', 'oldval')])}],
21836  'sysfs': [{'call': 'sysfs',
21837             'reason': set([('file_system_type', 'name'),
21838                            ('file_system_type', 'owner'),
21839                            ('filename', 'name')])},
```



```
21840             {'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
21841             {'call': 'swapoff', 'reason': set([('filename', 'name')])},
21842             {'call': 'openat', 'reason': set([('filename', 'name')])},
21843             {'call': 'uselib', 'reason': set([('filename', 'name')])},
21844             {'call': 'renameat2', 'reason': set([('filename', 'name')])},
21845             {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
21846             {'call': 'quotactl', 'reason': set([('filename', 'name')])},
21847             {'call': 'acct', 'reason': set([('filename', 'name')])},
21848             {'call': 'open', 'reason': set([('filename', 'name')])},
21849             {'call': 'unlink', 'reason': set([('filename', 'name')])},
21850             {'call': 'rmdir', 'reason': set([('filename', 'name')])},
21851             {'call': 'swapon', 'reason': set([('filename', 'name')])},
21852             {'call': 'mq_open', 'reason': set([('filename', 'name')])},
21853             {'call': 'unlinkat', 'reason': set([('filename', 'name')])}],
21854  'sysinfo': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
21855              {'call': 'rt_sigtimedwait',
21856               'reason': set([('mm_segment_t', 'seg'),
21857                              ('timespec', 'tv_nsec')])},
21858              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
21859              {'call': 'mq_unlink', 'reason': set([('timespec', 'tv_nsec')])},
21860              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
21861              {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
21862              {'call': 'sched_getaffinity',
21863               'reason': set([('mm_segment_t', 'seg')])},
21864              {'call': 'sched_setparam',
21865               'reason': set([('mm_segment_t', 'seg')])},
21866              {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
21867              {'call': 'memfd_create',
21868               'reason': set([('timespec', 'tv_nsec')])},
21869              {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
21870              {'call': 'readlinkat', 'reason': set([('timespec', 'tv_nsec')])},
21871              {'call': 'io_getevents',
21872               'reason': set([('timespec', 'tv_nsec')])},
21873              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
21874              {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
21875              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
21876              {'call': 'mq_timedreceive',
21877               'reason': set([('mm_segment_t', 'seg'),
21878                              ('timespec', 'tv_nsec')])},
21879              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
21880              {'call': 'utime', 'reason': set([('timespec', 'tv_nsec')])},
21881              {'call': 'sched_setaffinity',
21882               'reason': set([('mm_segment_t', 'seg')])},
21883              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
21884              {'call': 'semtimedop',
21885               'reason': set([('mm_segment_t', 'seg'),
21886                              ('timespec', 'tv_nsec')])},
21887              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
21888              {'call': 'settimeofday',
21889               'reason': set([('timespec', 'tv_nsec')])},
21890              {'call': 'sched_rr_get_interval',
21891               'reason': set([('mm_segment_t', 'seg'),
21892                              ('timespec', 'tv_nsec')])},
21893              {'call': 'timerfd_gettime',
21894               'reason': set([('timespec', 'tv_nsec')])},
21895              {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
21896              {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
21897              {'call': 'rt_sigprocmask',
21898               'reason': set([('mm_segment_t', 'seg')])},
21899              {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
21900              {'call': 'sigaltstack',
21901               'reason': set([('mm_segment_t', 'seg')])},
21902              {'call': 'sched_setattr',
21903               'reason': set([('mm_segment_t', 'seg')])},
21904              {'call': 'migrate_pages',
21905               'reason': set([('mm_segment_t', 'seg')])},
```

```
21906             {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
21907             {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
21908             {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
21909             {'call': 'inotify_add_watch',
21910              'reason': set([('timespec', 'tv_nsec')])},
21911             {'call': 'timer_settime',
21912              'reason': set([('timespec', 'tv_nsec')])},
21913             {'call': 'ftruncate', 'reason': set([('timespec', 'tv_nsec')])},
21914             {'call': 'timer_gettime',
21915              'reason': set([('timespec', 'tv_nsec')])},
21916             {'call': 'sysinfo',
21917              'reason': set([('sysinfo', 'mem_unit'),
21918                             ('sysinfo', 'totalram'),
21919                             ('sysinfo', 'totalswap')])},
21920             {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
21921             {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
21922             {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
21923             {'call': 'perf_event_open',
21924              'reason': set([('mm_segment_t', 'seg')])},
21925             {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
21926             {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
21927             {'call': 'rt_sigaction',
21928              'reason': set([('mm_segment_t', 'seg')])},
21929             {'call': 'futimesat', 'reason': set([('timespec', 'tv_nsec')])},
21930             {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
21931             {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
21932             {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
21933             {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
21934             {'call': 'getpriority',
21935              'reason': set([('mm_segment_t', 'seg')])},
21936             {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
21937             {'call': 'nanosleep', 'reason': set([('timespec', 'tv_nsec')])},
21938             {'call': 'mq_getsetattr',
21939              'reason': set([('timespec', 'tv_nsec')])},
21940             {'call': 'faccessat', 'reason': set([('timespec', 'tv_nsec')])},
21941             {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
21942             {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
21943             {'call': 'get_robust_list',
21944              'reason': set([('mm_segment_t', 'seg')])},
21945             {'call': 'mq_timedsend',
21946              'reason': set([('mm_segment_t', 'seg'),
21947                             ('timespec', 'tv_nsec')])},
21948             {'call': 'sched_getscheduler',
21949              'reason': set([('mm_segment_t', 'seg')])},
21950             {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
21951             {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
21952             {'call': 'epoll_wait', 'reason': set([('timespec', 'tv_nsec')])},
21953             {'call': 'sched_getattr',
21954              'reason': set([('mm_segment_t', 'seg')])},
21955             {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
21956             {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
21957             {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
21958             {'call': 'timerfd_settime',
21959              'reason': set([('timespec', 'tv_nsec')])},
21960             {'call': 'sched_setscheduler',
21961              'reason': set([('mm_segment_t', 'seg')])},
21962             {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
21963             {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
21964             {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
21965             {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
21966             {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
21967             {'call': 'setpriority',
21968              'reason': set([('mm_segment_t', 'seg')])},
21969             {'call': 'mq_notify', 'reason': set([('timespec', 'tv_nsec')])},
21970             {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
21971             {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
```



```
21972             {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
21973             {'call': 'clock_nanosleep',
21974              'reason': set([('timespec', 'tv_nsec')])},
21975             {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
21976             {'call': 'sched_getparam',
21977              'reason': set([('mm_segment_t', 'seg')])},
21978             {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
21979             {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
21980             {'call': 'sendfile64', 'reason': set([('timespec', 'tv_nsec')])},
21981             {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
21982  'syslog': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
21983             {'call': 'rt_sigtimedwait',
21984              'reason': set([('mm_segment_t', 'seg')])},
21985             {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
21986             {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
21987             {'call': 'sched_getaffinity',
21988              'reason': set([('mm_segment_t', 'seg')])},
21989             {'call': 'sched_setparam',
21990              'reason': set([('mm_segment_t', 'seg')])},
21991             {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
21992             {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
21993             {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
21994             {'call': 'mq_timedreceive',
21995              'reason': set([('mm_segment_t', 'seg')])},
21996             {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
21997             {'call': 'sched_setaffinity',
21998              'reason': set([('mm_segment_t', 'seg')])},
21999             {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
22000             {'call': 'semtimedop', 'reason': set([('mm_segment_t', 'seg')])},
22001             {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
22002             {'call': 'sched_rr_get_interval',
22003              'reason': set([('mm_segment_t', 'seg')])},
22004             {'call': 'rt_sigprocmask',
22005              'reason': set([('mm_segment_t', 'seg')])},
22006             {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
22007             {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
22008             {'call': 'sched_setattr',
22009              'reason': set([('mm_segment_t', 'seg')])},
22010             {'call': 'migrate_pages',
22011              'reason': set([('mm_segment_t', 'seg')])},
22012             {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
22013             {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
22014             {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
22015             {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
22016             {'call': 'perf_event_open',
22017              'reason': set([('mm_segment_t', 'seg')])},
22018             {'call': 'rt_sigaction',
22019              'reason': set([('mm_segment_t', 'seg')])},
22020             {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
22021             {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
22022             {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
22023             {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
22024             {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
22025             {'call': 'get_robust_list',
22026              'reason': set([('mm_segment_t', 'seg')])},
22027             {'call': 'mq_timedsend',
22028              'reason': set([('mm_segment_t', 'seg')])},
22029             {'call': 'sched_getscheduler',
22030              'reason': set([('mm_segment_t', 'seg')])},
22031             {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
22032             {'call': 'sched_getattr',
22033              'reason': set([('mm_segment_t', 'seg')])},
22034             {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
22035             {'call': 'sched_setscheduler',
22036              'reason': set([('mm_segment_t', 'seg')])},
22037             {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
```

```
            {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getparam',
             'reason': set([('mm_segment_t', 'seg')])}],
 'tee': [{'call': 'syncfs', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'vmsplice',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
         {'call': 'pwritev64',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
         {'call': 'fremovexattr',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'readahead',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'getdents',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'writev', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'preadv64',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'fchmod', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'pread64',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'signalfd4',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
         {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
         {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
         {'call': 'read', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'fchown', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'mq_timedreceive',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'utime', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'fsync', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'recvfrom',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'fsetxattr',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'sendto', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
         {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'sync_file_range',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'lseek', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'connect',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'getsockname',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'epoll_ctl',
          'reason': set([('fd', 'file'),
                         ('fd', 'flags'),
                         ('file', 'f_mode')])},
         {'call': 'flock',
          'reason': set([('fd', 'file'),
                         ('fd', 'flags'),
                         ('file', 'f_mode')])},
         {'call': 'pwritev',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'fchdir', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'openat', 'reason': set([('file', 'f_mode')])},
         {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
```



```
         {'call': 'accept4',
          'reason': set([('fd', 'file'),
                         ('fd', 'flags'),
                         ('file', 'f_mode')])},
         {'call': 'old_readdir',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'inotify_rm_watch',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
         {'call': 'utimensat',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'inotify_add_watch',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'preadv2',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'splice', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'ftruncate',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'preadv', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'getpeername',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
         {'call': 'setsockopt',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'socket', 'reason': set([('file', 'f_mode')])},
         {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
         {'call': 'fcntl', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'ioctl', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'pwrite64',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'perf_event_open',
          'reason': set([('fd', 'file'),
                         ('fd', 'flags'),
                         ('file', 'f_mode')])},
         {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
         {'call': 'pwritev64v2',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'futimesat',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'pwritev2',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'shutdown',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'acct', 'reason': set([('file', 'f_mode')])},
         {'call': 'open', 'reason': set([('file', 'f_mode')])},
         {'call': 'getsockopt',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'mq_getsetattr',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'dup', 'reason': set([('file', 'f_mode')])},
         {'call': 'fdatasync',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'setns', 'reason': set([('file', 'f_mode')])},
         {'call': 'getdents64',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'listen', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'copy_file_range',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'mq_timedsend',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'fgetxattr',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
         {'call': 'fcntl64',
          'reason': set([('fd', 'file'), ('fd', 'flags')])},
         {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
```

```
22170          {'call': 'fallocate',
22171           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22172          {'call': 'epoll_wait',
22173           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22174          {'call': 'llseek', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
22175          {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
22176          {'call': 'preadv64v2',
22177           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22178          {'call': 'readv', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
22179          {'call': 'fstatfs',
22180           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22181          {'call': 'fstatfs64',
22182           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22183          {'call': 'write', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
22184          {'call': 'mq_notify',
22185           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22186          {'call': 'sendfile',
22187           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22188          {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
22189          {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
22190          {'call': 'bind', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
22191          {'call': 'flistxattr',
22192           'reason': set([('fd', 'file'), ('fd', 'flags')])},
22193          {'call': 'sendfile64',
22194           'reason': set([('fd', 'file'), ('fd', 'flags')])}],
22195  'tgkill': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
22196             {'call': 'rt_sigtimedwait',
22197              'reason': set([('task_struct', 'cred')])},
22198             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
22199             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
22200             {'call': 'sched_getaffinity',
22201              'reason': set([('task_struct', 'cred')])},
22202             {'call': 'sched_setparam',
22203              'reason': set([('task_struct', 'cred')])},
22204             {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
22205             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
22206             {'call': 'mq_timedreceive',
22207              'reason': set([('task_struct', 'cred')])},
22208             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
22209             {'call': 'sched_setaffinity',
22210              'reason': set([('task_struct', 'cred')])},
22211             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
22212             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
22213             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
22214             {'call': 'sched_rr_get_interval',
22215              'reason': set([('task_struct', 'cred')])},
22216             {'call': 'rt_sigprocmask',
22217              'reason': set([('task_struct', 'cred')])},
22218             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
22219             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
22220             {'call': 'sched_setattr',
22221              'reason': set([('task_struct', 'cred')])},
22222             {'call': 'migrate_pages',
22223              'reason': set([('task_struct', 'cred')])},
22224             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
22225             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
22226             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
22227             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
22228             {'call': 'perf_event_open',
22229              'reason': set([('task_struct', 'cred')])},
22230             {'call': 'rt_sigaction',
22231              'reason': set([('task_struct', 'cred')])},
22232             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
22233             {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
22234             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
22235             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
```



```
22236             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
22237             {'call': 'get_robust_list',
22238              'reason': set([('task_struct', 'cred')])},
22239             {'call': 'mq_timedsend',
22240              'reason': set([('task_struct', 'cred')])},
22241             {'call': 'sched_getscheduler',
22242              'reason': set([('task_struct', 'cred')])},
22243             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
22244             {'call': 'sched_getattr',
22245              'reason': set([('task_struct', 'cred')])},
22246             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
22247             {'call': 'sched_setscheduler',
22248              'reason': set([('task_struct', 'cred')])},
22249             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
22250             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
22251             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
22252             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
22253             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
22254             {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
22255             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
22256             {'call': 'sched_getparam',
22257              'reason': set([('task_struct', 'cred')])}],
22258  'timer_create': [{'call': 'clock_getres',
22259                    'reason': set([('k_clock', 'timer_create')])},
22260                   {'call': 'timer_delete',
22261                    'reason': set([('k_clock', 'timer_create'),
22262                                   ('k_itimer', 'it_pid')])},
22263                   {'call': 'timer_create',
22264                    'reason': set([('k_clock', 'timer_create'),
22265                                   ('k_itimer', 'it_pid')])},
22266                   {'call': 'clock_gettime',
22267                    'reason': set([('k_clock', 'timer_create')])},
22268                   {'call': 'timer_settime',
22269                    'reason': set([('k_clock', 'timer_create'),
22270                                   ('k_itimer', 'it_pid')])},
22271                   {'call': 'timer_gettime',
22272                    'reason': set([('k_clock', 'timer_create'),
22273                                   ('k_itimer', 'it_pid')])},
22274                   {'call': 'clock_settime',
22275                    'reason': set([('k_clock', 'timer_create')])},
22276                   {'call': 'timer_getoverrun',
22277                    'reason': set([('k_itimer', 'it_pid')])},
22278                   {'call': 'clock_nanosleep',
22279                    'reason': set([('k_clock', 'timer_create')])},
22280                   {'call': 'clock_adjtime',
22281                    'reason': set([('k_clock', 'timer_create')])}],
22282  'timer_delete': [{'call': 'keyctl',
22283                    'reason': set([('task_struct', 'signal')])},
22284                   {'call': 'rt_sigtimedwait',
22285                    'reason': set([('task_struct', 'signal')])},
22286                   {'call': 'msgrcv',
22287                    'reason': set([('task_struct', 'signal')])},
22288                   {'call': 'kill',
22289                    'reason': set([('task_struct', 'signal')])},
22290                   {'call': 'clock_getres',
22291                    'reason': set([('k_clock', 'timer_del')])},
22292                   {'call': 'timer_delete',
22293                    'reason': set([('k_clock', 'timer_del'),
22294                                   ('k_itimer', 'it_pid'),
22295                                   ('k_itimer', 'it_signal'),
22296                                   ('k_itimer', 'sigq')])},
22297                   {'call': 'sched_getaffinity',
22298                    'reason': set([('task_struct', 'signal')])},
22299                   {'call': 'sched_setparam',
22300                    'reason': set([('task_struct', 'signal')])},
22301                   {'call': 'ioprio_set',
```

```
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'capget',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'signal',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'umount',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'timer_create',
                 'reason': set([('k_clock', 'timer_del'),
                                ('k_itimer', 'it_pid'),
                                ('k_itimer', 'it_signal'),
                                ('k_itimer', 'sigq')])},
                {'call': 'clock_gettime',
                 'reason': set([('k_clock', 'timer_del')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'timer_settime',
                 'reason': set([('k_clock', 'timer_del'),
                                ('k_itimer', 'it_pid'),
                                ('k_itimer', 'it_signal'),
                                ('k_itimer', 'sigq')])},
                {'call': 'timer_gettime',
                 'reason': set([('k_clock', 'timer_del'),
                                ('k_itimer', 'it_pid'),
                                ('k_itimer', 'it_signal'),
                                ('k_itimer', 'sigq')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'clock_settime',
                 'reason': set([('k_clock', 'timer_del')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'signal')])},
```



```
                {'call': 'fork',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'timer_getoverrun',
                 'reason': set([('k_itimer', 'it_pid'),
                                ('k_itimer', 'it_signal'),
                                ('k_itimer', 'sigq')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'clock_nanosleep',
                 'reason': set([('k_clock', 'timer_del')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'signal')])},
                {'call': 'clock_adjtime',
                 'reason': set([('k_clock', 'timer_del')])}],
 'timer_getoverrun': [{'call': 'keyctl',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'rt_sigtimedwait',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'msgrcv',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'kill',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'timer_delete',
                       'reason': set([('k_itimer', 'it_signal')])},
                      {'call': 'sched_getaffinity',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_setparam',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'ioprio_set',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'getppid',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'mq_timedreceive',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'capget',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_setaffinity',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'signal',
                       'reason': set([('task_struct', 'signal')])},
```

```
                    {'call': 'semtimedop',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'umount',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'timer_create',
                     'reason': set([('k_itimer', 'it_signal')])},
                    {'call': 'sched_rr_get_interval',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'rt_sigprocmask',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'setsid',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sigaltstack',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sched_setattr',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'migrate_pages',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'getitimer',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'setpgid',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'timer_settime',
                     'reason': set([('k_itimer', 'it_signal')])},
                    {'call': 'timer_gettime',
                     'reason': set([('k_itimer', 'it_signal')])},
                    {'call': 'getsid',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'prlimit64',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'perf_event_open',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'rt_sigaction',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'getpgid',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'getpriority',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sigaction',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'setns',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'fork',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'get_robust_list',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'ptrace',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sched_getattr',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'getrusage',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'setitimer',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'ioprio_get',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'vfork',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'prctl',
                     'reason': set([('task_struct', 'signal')])},
```



```
                    {'call': 'move_pages',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'setpriority',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'timer_getoverrun',
                     'reason': set([('k_itimer', 'it_signal')])},
                    {'call': 'clone',
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sched_getparam',
                     'reason': set([('task_struct', 'signal')])}],
 'timer_gettime': [{'call': 'clock_getres',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'timer_delete',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'timer_create',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_gettime',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'timer_settime',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'timer_gettime',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_settime',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_nanosleep',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_adjtime',
                    'reason': set([('k_clock', 'timer_get')])}],
 'timer_settime': [{'call': 'clock_getres',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'timer_delete',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'timer_create',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_gettime',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'timer_settime',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'timer_gettime',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_settime',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_nanosleep',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_adjtime',
                    'reason': set([('k_clock', 'timer_set')])}],
 'timerfd_create': [{'call': 'timerfd_gettime',
                     'reason': set([('timerfd_ctx', 'clockid')])},
                    {'call': 'timerfd_settime',
                     'reason': set([('timerfd_ctx', 'clockid')])},
                    {'call': 'timerfd_create',
                     'reason': set([('timerfd_ctx', 'clockid')])}],
 'timerfd_gettime': [{'call': 'timerfd_gettime',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])},
                     {'call': 'timerfd_settime',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])},
                     {'call': 'timerfd_create',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])}],
 'timerfd_settime': [{'call': 'timerfd_gettime',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])},
                     {'call': 'timerfd_settime',
                      'reason': set([('timerfd_ctx', 'expired'),
```

```
                                    ('timerfd_ctx', 'tintv')])},
                    {'call': 'timerfd_create',
                     'reason': set([('timerfd_ctx', 'expired'),
                                    ('timerfd_ctx', 'tintv')])}],
 'tkill': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
           {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getaffinity',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setparam',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'mq_timedreceive',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setaffinity',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
           {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
           {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_rr_get_interval',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'rt_sigprocmask',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setattr',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'migrate_pages',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
           {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
           {'call': 'perf_event_open',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'rt_sigaction', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
           {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
           {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
           {'call': 'get_robust_list',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'mq_timedsend', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getscheduler',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getattr',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setscheduler',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
           {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
           {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
           {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
           {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
           {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
           {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getparam',
            'reason': set([('task_struct', 'cred')])}],
 'umount': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
```



```
            {'call': 'keyctl', 'reason': set([('task_struct', 'flags')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'msgrcv', 'reason': set([('task_struct', 'flags')])},
            {'call': 'eventfd2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mq_unlink',
             'reason': set([('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'kill', 'reason': set([('task_struct', 'flags')])},
            {'call': 'swapoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'pivot_root',
             'reason': set([('mount', 'mnt_ns'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'memfd_create',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'ioprio_set', 'reason': set([('task_struct', 'flags')])},
            {'call': 'remap_file_pages',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'dup3',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'getppid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'capget', 'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setaffinity',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
            {'call': 'unshare',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'signal', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'semtimedop', 'reason': set([('task_struct', 'flags')])},
            {'call': 'umount',
             'reason': set([('mount', 'mnt_ns'),
                            ('super_block', 's_flags'),
                            ('task_struct', 'flags'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'epoll_create1',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'epoll_ctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'flock',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'openat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'lookup_dcookie',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'uselib',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'accept4',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'setsid', 'reason': set([('task_struct', 'flags')])},
```

```
            {'call': 'sigaltstack',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'socketpair',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'getitimer', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setpgid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'getcwd',
             'reason': set([('mount', 'mnt_ns'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'getsid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'shmat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'socket',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'pipe2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'prlimit64', 'reason': set([('task_struct', 'flags')])},
            {'call': 'perf_event_open',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('task_struct', 'flags')])},
            {'call': 'shmdt',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'quotactl',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('super_block', 's_flags')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'getpgid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'acct',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'open',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'getpriority',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sigaction', 'reason': set([('task_struct', 'flags')])},
            {'call': 'dup',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'setns',
             'reason': set([('nsproxy', 'mnt_ns'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('task_struct', 'flags')])},
            {'call': 'fork', 'reason': set([('task_struct', 'flags')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'ptrace', 'reason': set([('task_struct', 'flags')])},
            {'call': 'shmctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'swapon',
             'reason': set([('path', 'dentry'),
```



```
                            ('path', 'mnt'),
                            ('super_block', 's_flags')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'getrusage', 'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setitimer', 'reason': set([('task_struct', 'flags')])},
            {'call': 'ioprio_get', 'reason': set([('task_struct', 'flags')])},
            {'call': 'vfork', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'mmap_pgoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'prctl', 'reason': set([('task_struct', 'flags')])},
            {'call': 'move_pages', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setpriority',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'clone', 'reason': set([('task_struct', 'flags')])},
            {'call': 'mq_open',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'sched_getparam',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'open_by_handle_at',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'uname': [{'call': 'keyctl',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'msgrcv',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_getaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setparam',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'ioprio_set',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'personality',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getppid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'mq_timedreceive',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'capget',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'signal',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'semtimedop',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'umount',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_rr_get_interval',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigprocmask',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'setsid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sigaltstack',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setattr',
```

```
22830             'reason': set([('task_struct', 'personality')])},
22831            {'call': 'migrate_pages',
22832             'reason': set([('task_struct', 'personality')])},
22833            {'call': 'getitimer',
22834             'reason': set([('task_struct', 'personality')])},
22835            {'call': 'setpgid',
22836             'reason': set([('task_struct', 'personality')])},
22837            {'call': 'getsid',
22838             'reason': set([('task_struct', 'personality')])},
22839            {'call': 'prlimit64',
22840             'reason': set([('task_struct', 'personality')])},
22841            {'call': 'perf_event_open',
22842             'reason': set([('task_struct', 'personality')])},
22843            {'call': 'rt_sigaction',
22844             'reason': set([('task_struct', 'personality')])},
22845            {'call': 'getpgid',
22846             'reason': set([('task_struct', 'personality')])},
22847            {'call': 'getpriority',
22848             'reason': set([('task_struct', 'personality')])},
22849            {'call': 'sigaction',
22850             'reason': set([('task_struct', 'personality')])},
22851            {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
22852            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
22853            {'call': 'get_robust_list',
22854             'reason': set([('task_struct', 'personality')])},
22855            {'call': 'mq_timedsend',
22856             'reason': set([('task_struct', 'personality')])},
22857            {'call': 'sched_getscheduler',
22858             'reason': set([('task_struct', 'personality')])},
22859            {'call': 'ptrace',
22860             'reason': set([('task_struct', 'personality')])},
22861            {'call': 'sched_getattr',
22862             'reason': set([('task_struct', 'personality')])},
22863            {'call': 'getrusage',
22864             'reason': set([('task_struct', 'personality')])},
22865            {'call': 'sched_setscheduler',
22866             'reason': set([('task_struct', 'personality')])},
22867            {'call': 'setitimer',
22868             'reason': set([('task_struct', 'personality')])},
22869            {'call': 'ioprio_get',
22870             'reason': set([('task_struct', 'personality')])},
22871            {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
22872            {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
22873            {'call': 'move_pages',
22874             'reason': set([('task_struct', 'personality')])},
22875            {'call': 'setpriority',
22876             'reason': set([('task_struct', 'personality')])},
22877            {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
22878            {'call': 'sched_getparam',
22879             'reason': set([('task_struct', 'personality')])}],
22880  'unlink': [{'call': 'eventfd2',
22881              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22882             {'call': 'mq_unlink', 'reason': set([('dentry', 'd_inode')])},
22883             {'call': 'swapoff',
22884              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22885             {'call': 'pivot_root',
22886              'reason': set([('dentry', 'd_inode'),
22887                             ('path', 'dentry'),
22888                             ('path', 'mnt')])},
22889             {'call': 'memfd_create',
22890              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22891             {'call': 'remap_file_pages',
22892              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22893             {'call': 'dup3',
22894              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22895             {'call': 'unshare',
```



```
22896              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22897             {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
22898             {'call': 'epoll_create1',
22899              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22900             {'call': 'epoll_ctl',
22901              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22902             {'call': 'flock',
22903              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22904             {'call': 'openat',
22905              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22906             {'call': 'lookup_dcookie',
22907              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22908             {'call': 'uselib',
22909              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22910             {'call': 'renameat2', 'reason': set([('dentry', 'd_inode')])},
22911             {'call': 'accept4',
22912              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22913             {'call': 'socketpair',
22914              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22915             {'call': 'getcwd',
22916              'reason': set([('dentry', 'd_inode'),
22917                             ('path', 'dentry'),
22918                             ('path', 'mnt')])},
22919             {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
22920             {'call': 'shmat',
22921              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22922             {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
22923             {'call': 'socket',
22924              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22925             {'call': 'symlinkat', 'reason': set([('dentry', 'd_inode')])},
22926             {'call': 'pipe2',
22927              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22928             {'call': 'perf_event_open',
22929              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22930             {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
22931             {'call': 'shmdt',
22932              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22933             {'call': 'quotactl',
22934              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22935             {'call': 'acct',
22936              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22937             {'call': 'open',
22938              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22939             {'call': 'unlink', 'reason': set([('dentry', 'd_inode')])},
22940             {'call': 'rmdir', 'reason': set([('dentry', 'd_inode')])},
22941             {'call': 'dup',
22942              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22943             {'call': 'setns',
22944              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22945             {'call': 'shmctl',
22946              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22947             {'call': 'swapon',
22948              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22949             {'call': 'mmap_pgoff',
22950              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22951             {'call': 'mq_open',
22952              'reason': set([('dentry', 'd_inode'),
22953                             ('path', 'dentry'),
22954                             ('path', 'mnt')])},
22955             {'call': 'unlinkat', 'reason': set([('dentry', 'd_inode')])},
22956             {'call': 'open_by_handle_at',
22957              'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
22958  'unlinkat': [{'call': 'eventfd2',
22959                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
22960               {'call': 'mq_unlink', 'reason': set([('dentry', 'd_inode')])},
22961               {'call': 'swapoff',
```

```
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'pivot_root',
                'reason': set([('dentry', 'd_inode'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'memfd_create',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'remap_file_pages',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'dup3',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'unshare',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'epoll_create1',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'epoll_ctl',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'flock',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'openat',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'lookup_dcookie',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'uselib',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'renameat2', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'accept4',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'socketpair',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'getcwd',
                'reason': set([('dentry', 'd_inode'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'shmat',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'socket',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'symlinkat', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'pipe2',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'perf_event_open',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'shmdt',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'quotactl',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'acct',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'open',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'unlink', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'rmdir', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'dup',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'setns',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'shmctl',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'swapon',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mmap_pgoff',
```



```
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mq_open',
                'reason': set([('dentry', 'd_inode'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'unlinkat', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'open_by_handle_at',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'unshare': [{'call': 'unshare', 'reason': set([('fs_struct', 'users')])}],
 'uselib': [{'call': 'syncfs', 'reason': set([('super_block', 's_iflags')])},
            {'call': 'mq_unlink', 'reason': set([('vfsmount', 'mnt_flags')])},
            {'call': 'pivot_root',
             'reason': set([('vfsmount', 'mnt_flags')])},
            {'call': 'ustat', 'reason': set([('super_block', 's_iflags')])},
            {'call': 'umount',
             'reason': set([('super_block', 's_iflags'),
                            ('vfsmount', 'mnt_flags')])},
            {'call': 'uselib',
             'reason': set([('linux_binfmt', 'load_shlib'),
                            ('linux_binfmt', 'module')])},
            {'call': 'getcwd', 'reason': set([('vfsmount', 'mnt_flags')])},
            {'call': 'quotactl',
             'reason': set([('super_block', 's_iflags')])},
            {'call': 'acct', 'reason': set([('vfsmount', 'mnt_flags')])},
            {'call': 'swapon', 'reason': set([('super_block', 's_iflags')])},
            {'call': 'mq_open', 'reason': set([('vfsmount', 'mnt_flags')])}],
 'ustat': [{'call': 'syncfs', 'reason': set([('super_block', 's_root')])},
           {'call': 'ustat', 'reason': set([('super_block', 's_root')])},
           {'call': 'umount', 'reason': set([('super_block', 's_root')])},
           {'call': 'quotactl', 'reason': set([('super_block', 's_root')])},
           {'call': 'swapon', 'reason': set([('super_block', 's_root')])}],
 'utime': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
           {'call': 'mq_unlink', 'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
           {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
           {'call': 'readahead', 'reason': set([('fd', 'file')])},
           {'call': 'getdents', 'reason': set([('fd', 'file')])},
           {'call': 'writev', 'reason': set([('fd', 'file')])},
           {'call': 'preadv64', 'reason': set([('fd', 'file')])},
           {'call': 'fchmod',
            'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
           {'call': 'pread64', 'reason': set([('fd', 'file')])},
           {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
           {'call': 'memfd_create', 'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'readlinkat', 'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'read', 'reason': set([('fd', 'file')])},
           {'call': 'io_getevents', 'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'fchown',
            'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
           {'call': 'mq_timedreceive',
            'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
           {'call': 'utime',
            'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
           {'call': 'fsync', 'reason': set([('fd', 'file')])},
           {'call': 'bpf', 'reason': set([('fd', 'file')])},
           {'call': 'semtimedop', 'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
           {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
           {'call': 'settimeofday', 'reason': set([('timespec', 'tv_nsec')])},
           {'call': 'sendto', 'reason': set([('fd', 'file')])},
           {'call': 'sched_rr_get_interval',
            'reason': set([('timespec', 'tv_nsec')])},
```

```
23094             {'call': 'timerfd_gettime',
23095              'reason': set([('timespec', 'tv_nsec')])},
23096             {'call': 'tee', 'reason': set([('fd', 'file')])},
23097             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
23098             {'call': 'lseek', 'reason': set([('fd', 'file')])},
23099             {'call': 'connect', 'reason': set([('fd', 'file')])},
23100             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
23101             {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
23102             {'call': 'flock', 'reason': set([('fd', 'file')])},
23103             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
23104             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
23105             {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
23106             {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
23107             {'call': 'accept4', 'reason': set([('fd', 'file')])},
23108             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
23109             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
23110             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
23111             {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
23112             {'call': 'inotify_add_watch',
23113              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23114             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
23115             {'call': 'timer_settime',
23116              'reason': set([('timespec', 'tv_nsec')])},
23117             {'call': 'splice', 'reason': set([('fd', 'file')])},
23118             {'call': 'ftruncate',
23119              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23120             {'call': 'timer_gettime',
23121              'reason': set([('timespec', 'tv_nsec')])},
23122             {'call': 'preadv', 'reason': set([('fd', 'file')])},
23123             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
23124             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
23125             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
23126             {'call': 'ioctl',
23127              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23128             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
23129             {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
23130             {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
23131             {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
23132             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
23133             {'call': 'futimesat',
23134              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23135             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
23136             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
23137             {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
23138             {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
23139             {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
23140             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
23141             {'call': 'nanosleep', 'reason': set([('timespec', 'tv_nsec')])},
23142             {'call': 'mq_getsetattr',
23143              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23144             {'call': 'faccessat', 'reason': set([('timespec', 'tv_nsec')])},
23145             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
23146             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
23147             {'call': 'listen', 'reason': set([('fd', 'file')])},
23148             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
23149             {'call': 'mq_timedsend',
23150              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23151             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
23152             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
23153             {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
23154             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
23155             {'call': 'epoll_wait',
23156              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23157             {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
23158             {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
23159             {'call': 'timerfd_settime',
```



```
23160              'reason': set([('timespec', 'tv_nsec')])},
23161             {'call': 'llseek', 'reason': set([('fd', 'file')])},
23162             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
23163             {'call': 'readv', 'reason': set([('fd', 'file')])},
23164             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
23165             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
23166             {'call': 'write', 'reason': set([('fd', 'file')])},
23167             {'call': 'mq_notify',
23168              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23169             {'call': 'sendfile',
23170              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23171             {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
23172             {'call': 'clock_nanosleep',
23173              'reason': set([('timespec', 'tv_nsec')])},
23174             {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
23175             {'call': 'bind', 'reason': set([('fd', 'file')])},
23176             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
23177             {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
23178             {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
23179             {'call': 'sendfile64',
23180              'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23181             {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
23182  'utimensat': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
23183                {'call': 'rt_sigtimedwait',
23184                 'reason': set([('timespec', 'tv_nsec')])},
23185                {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
23186                {'call': 'mq_unlink',
23187                 'reason': set([('timespec', 'tv_nsec')])},
23188                {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
23189                {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
23190                {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
23191                {'call': 'readahead', 'reason': set([('fd', 'file')])},
23192                {'call': 'getdents', 'reason': set([('fd', 'file')])},
23193                {'call': 'writev', 'reason': set([('fd', 'file')])},
23194                {'call': 'preadv64', 'reason': set([('fd', 'file')])},
23195                {'call': 'fchmod',
23196                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23197                {'call': 'pread64', 'reason': set([('fd', 'file')])},
23198                {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
23199                {'call': 'memfd_create',
23200                 'reason': set([('timespec', 'tv_nsec')])},
23201                {'call': 'readlinkat',
23202                 'reason': set([('timespec', 'tv_nsec')])},
23203                {'call': 'read', 'reason': set([('fd', 'file')])},
23204                {'call': 'io_getevents',
23205                 'reason': set([('timespec', 'tv_nsec')])},
23206                {'call': 'fchown',
23207                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23208                {'call': 'mq_timedreceive',
23209                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23210                {'call': 'utime',
23211                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23212                {'call': 'fsync', 'reason': set([('fd', 'file')])},
23213                {'call': 'bpf', 'reason': set([('fd', 'file')])},
23214                {'call': 'semtimedop',
23215                 'reason': set([('timespec', 'tv_nsec')])},
23216                {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
23217                {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
23218                {'call': 'settimeofday',
23219                 'reason': set([('timespec', 'tv_nsec')])},
23220                {'call': 'sendto', 'reason': set([('fd', 'file')])},
23221                {'call': 'sched_rr_get_interval',
23222                 'reason': set([('timespec', 'tv_nsec')])},
23223                {'call': 'timerfd_gettime',
23224                 'reason': set([('timespec', 'tv_nsec')])},
23225                {'call': 'tee', 'reason': set([('fd', 'file')])},
```

```
23226                {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
23227                {'call': 'lseek', 'reason': set([('fd', 'file')])},
23228                {'call': 'connect', 'reason': set([('fd', 'file')])},
23229                {'call': 'getsockname', 'reason': set([('fd', 'file')])},
23230                {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
23231                {'call': 'flock', 'reason': set([('fd', 'file')])},
23232                {'call': 'pwritev', 'reason': set([('fd', 'file')])},
23233                {'call': 'fchdir', 'reason': set([('fd', 'file')])},
23234                {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
23235                {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
23236                {'call': 'accept4', 'reason': set([('fd', 'file')])},
23237                {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
23238                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
23239                {'call': 'utimensat', 'reason': set([('fd', 'file')])},
23240                {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
23241                {'call': 'inotify_add_watch',
23242                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23243                {'call': 'preadv2', 'reason': set([('fd', 'file')])},
23244                {'call': 'timer_settime',
23245                 'reason': set([('timespec', 'tv_nsec')])},
23246                {'call': 'splice', 'reason': set([('fd', 'file')])},
23247                {'call': 'ftruncate',
23248                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23249                {'call': 'timer_gettime',
23250                 'reason': set([('timespec', 'tv_nsec')])},
23251                {'call': 'preadv', 'reason': set([('fd', 'file')])},
23252                {'call': 'getpeername', 'reason': set([('fd', 'file')])},
23253                {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
23254                {'call': 'fcntl', 'reason': set([('fd', 'file')])},
23255                {'call': 'ioctl',
23256                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23257                {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
23258                {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
23259                {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
23260                {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
23261                {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
23262                {'call': 'futimesat',
23263                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23264                {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
23265                {'call': 'shutdown', 'reason': set([('fd', 'file')])},
23266                {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
23267                {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
23268                {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
23269                {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
23270                {'call': 'nanosleep',
23271                 'reason': set([('timespec', 'tv_nsec')])},
23272                {'call': 'mq_getsetattr',
23273                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23274                {'call': 'faccessat',
23275                 'reason': set([('timespec', 'tv_nsec')])},
23276                {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
23277                {'call': 'getdents64', 'reason': set([('fd', 'file')])},
23278                {'call': 'listen', 'reason': set([('fd', 'file')])},
23279                {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
23280                {'call': 'mq_timedsend',
23281                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23282                {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
23283                {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
23284                {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
23285                {'call': 'fallocate', 'reason': set([('fd', 'file')])},
23286                {'call': 'epoll_wait',
23287                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23288                {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
23289                {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
23290                {'call': 'timerfd_settime',
23291                 'reason': set([('timespec', 'tv_nsec')])},
```



```
23292                {'call': 'llseek', 'reason': set([('fd', 'file')])},
23293                {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
23294                {'call': 'readv', 'reason': set([('fd', 'file')])},
23295                {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
23296                {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
23297                {'call': 'write', 'reason': set([('fd', 'file')])},
23298                {'call': 'mq_notify',
23299                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23300                {'call': 'sendfile',
23301                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23302                {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
23303                {'call': 'clock_nanosleep',
23304                 'reason': set([('timespec', 'tv_nsec')])},
23305                {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
23306                {'call': 'bind', 'reason': set([('fd', 'file')])},
23307                {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
23308                {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
23309                {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
23310                {'call': 'sendfile64',
23311                 'reason': set([('fd', 'file'), ('timespec', 'tv_nsec')])},
23312                {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
23313  'vmsplice': [{'call': 'syncfs',
23314                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23315               {'call': 'vmsplice',
23316                'reason': set([('fd', 'file'),
23317                               ('fd', 'flags'),
23318                               ('splice_desc', 'total_len')])},
23319               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
23320               {'call': 'pwritev64',
23321                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23322               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
23323               {'call': 'fremovexattr',
23324                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23325               {'call': 'readahead',
23326                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23327               {'call': 'getdents',
23328                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23329               {'call': 'writev',
23330                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23331               {'call': 'preadv64',
23332                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23333               {'call': 'fchmod',
23334                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23335               {'call': 'pread64',
23336                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23337               {'call': 'signalfd4',
23338                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23339               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
23340               {'call': 'remap_file_pages',
23341                'reason': set([('file', 'f_mode')])},
23342               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
23343               {'call': 'read',
23344                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23345               {'call': 'fchown',
23346                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23347               {'call': 'mq_timedreceive',
23348                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23349               {'call': 'utime',
23350                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23351               {'call': 'fsync',
23352                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23353               {'call': 'bpf',
23354                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23355               {'call': 'recvfrom',
23356                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23357               {'call': 'fsetxattr',
```

```
23358                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23359               {'call': 'sendto',
23360                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23361               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
23362               {'call': 'tee',
23363                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23364               {'call': 'sync_file_range',
23365                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23366               {'call': 'lseek',
23367                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23368               {'call': 'connect',
23369                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23370               {'call': 'getsockname',
23371                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23372               {'call': 'epoll_ctl',
23373                'reason': set([('fd', 'file'),
23374                               ('fd', 'flags'),
23375                               ('file', 'f_mode')])},
23376               {'call': 'flock',
23377                'reason': set([('fd', 'file'),
23378                               ('fd', 'flags'),
23379                               ('file', 'f_mode')])},
23380               {'call': 'pwritev',
23381                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23382               {'call': 'fchdir',
23383                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23384               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
23385               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
23386               {'call': 'accept4',
23387                'reason': set([('fd', 'file'),
23388                               ('fd', 'flags'),
23389                               ('file', 'f_mode')])},
23390               {'call': 'old_readdir',
23391                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23392               {'call': 'inotify_rm_watch',
23393                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23394               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
23395               {'call': 'utimensat',
23396                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23397               {'call': 'inotify_add_watch',
23398                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23399               {'call': 'preadv2',
23400                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23401               {'call': 'splice',
23402                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23403               {'call': 'ftruncate',
23404                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23405               {'call': 'preadv',
23406                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23407               {'call': 'getpeername',
23408                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23409               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
23410               {'call': 'setsockopt',
23411                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23412               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
23413               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
23414               {'call': 'fcntl',
23415                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23416               {'call': 'ioctl',
23417                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23418               {'call': 'pwrite64',
23419                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23420               {'call': 'perf_event_open',
23421                'reason': set([('fd', 'file'),
23422                               ('fd', 'flags'),
23423                               ('file', 'f_mode')])},
```



```
23424               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
23425               {'call': 'pwritev64v2',
23426                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23427               {'call': 'futimesat',
23428                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23429               {'call': 'pwritev2',
23430                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23431               {'call': 'shutdown',
23432                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23433               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
23434               {'call': 'open', 'reason': set([('file', 'f_mode')])},
23435               {'call': 'getsockopt',
23436                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23437               {'call': 'mq_getsetattr',
23438                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23439               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
23440               {'call': 'fdatasync',
23441                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23442               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
23443               {'call': 'getdents64',
23444                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23445               {'call': 'listen',
23446                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23447               {'call': 'copy_file_range',
23448                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23449               {'call': 'mq_timedsend',
23450                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23451               {'call': 'fgetxattr',
23452                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23453               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
23454               {'call': 'fcntl64',
23455                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23456               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
23457               {'call': 'fallocate',
23458                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23459               {'call': 'epoll_wait',
23460                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23461               {'call': 'llseek',
23462                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23463               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
23464               {'call': 'preadv64v2',
23465                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23466               {'call': 'readv',
23467                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23468               {'call': 'fstatfs',
23469                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23470               {'call': 'fstatfs64',
23471                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23472               {'call': 'write',
23473                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23474               {'call': 'mq_notify',
23475                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23476               {'call': 'sendfile',
23477                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23478               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
23479               {'call': 'open_by_handle_at',
23480                'reason': set([('file', 'f_mode')])},
23481               {'call': 'bind',
23482                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23483               {'call': 'flistxattr',
23484                'reason': set([('fd', 'file'), ('fd', 'flags')])},
23485               {'call': 'sendfile64',
23486                'reason': set([('fd', 'file'), ('fd', 'flags')])}],
23487  'write': [{'call': 'syncfs',
23488             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23489            {'call': 'vmsplice',
```

```
23490             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23491            {'call': 'pwritev64',
23492             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23493            {'call': 'fremovexattr',
23494             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23495            {'call': 'readahead',
23496             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23497            {'call': 'getdents',
23498             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23499            {'call': 'writev',
23500             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23501            {'call': 'preadv64',
23502             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23503            {'call': 'fchmod',
23504             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23505            {'call': 'pread64',
23506             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23507            {'call': 'signalfd4',
23508             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23509            {'call': 'read', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
23510            {'call': 'fchown',
23511             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23512            {'call': 'mq_timedreceive',
23513             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23514            {'call': 'utime',
23515             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23516            {'call': 'fsync',
23517             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23518            {'call': 'bpf', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
23519            {'call': 'recvfrom',
23520             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23521            {'call': 'fsetxattr',
23522             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23523            {'call': 'sendto',
23524             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23525            {'call': 'tee', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
23526            {'call': 'sync_file_range',
23527             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23528            {'call': 'lseek',
23529             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23530            {'call': 'connect',
23531             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23532            {'call': 'getsockname',
23533             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23534            {'call': 'epoll_ctl',
23535             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23536            {'call': 'flock',
23537             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23538            {'call': 'pwritev',
23539             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23540            {'call': 'fchdir',
23541             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23542            {'call': 'accept4',
23543             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23544            {'call': 'old_readdir',
23545             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23546            {'call': 'inotify_rm_watch',
23547             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23548            {'call': 'utimensat',
23549             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23550            {'call': 'inotify_add_watch',
23551             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23552            {'call': 'preadv2',
23553             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23554            {'call': 'splice',
23555             'reason': set([('fd', 'file'), ('fd', 'flags')])},
```



```
23556            {'call': 'ftruncate',
23557             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23558            {'call': 'preadv',
23559             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23560            {'call': 'getpeername',
23561             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23562            {'call': 'setsockopt',
23563             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23564            {'call': 'fcntl',
23565             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23566            {'call': 'ioctl',
23567             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23568            {'call': 'pwrite64',
23569             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23570            {'call': 'perf_event_open',
23571             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23572            {'call': 'pwritev64v2',
23573             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23574            {'call': 'futimesat',
23575             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23576            {'call': 'pwritev2',
23577             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23578            {'call': 'shutdown',
23579             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23580            {'call': 'getsockopt',
23581             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23582            {'call': 'mq_getsetattr',
23583             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23584            {'call': 'fdatasync',
23585             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23586            {'call': 'getdents64',
23587             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23588            {'call': 'listen',
23589             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23590            {'call': 'copy_file_range',
23591             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23592            {'call': 'mq_timedsend',
23593             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23594            {'call': 'fgetxattr',
23595             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23596            {'call': 'fcntl64',
23597             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23598            {'call': 'fallocate',
23599             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23600            {'call': 'epoll_wait',
23601             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23602            {'call': 'llseek',
23603             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23604            {'call': 'preadv64v2',
23605             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23606            {'call': 'readv',
23607             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23608            {'call': 'fstatfs',
23609             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23610            {'call': 'fstatfs64',
23611             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23612            {'call': 'write',
23613             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23614            {'call': 'mq_notify',
23615             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23616            {'call': 'sendfile',
23617             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23618            {'call': 'bind', 'reason': set([('fd', 'file'), ('fd', 'flags')])},
23619            {'call': 'flistxattr',
23620             'reason': set([('fd', 'file'), ('fd', 'flags')])},
23621            {'call': 'sendfile64',
```

```
23622             'reason': set([('fd', 'file'), ('fd', 'flags')])}],
23623  'writev': [{'call': 'syncfs', 'reason': set([('fd', 'file')])},
23624             {'call': 'vmsplice', 'reason': set([('fd', 'file')])},
23625             {'call': 'pwritev64', 'reason': set([('fd', 'file')])},
23626             {'call': 'fremovexattr', 'reason': set([('fd', 'file')])},
23627             {'call': 'readahead', 'reason': set([('fd', 'file')])},
23628             {'call': 'getdents', 'reason': set([('fd', 'file')])},
23629             {'call': 'writev', 'reason': set([('fd', 'file')])},
23630             {'call': 'preadv64', 'reason': set([('fd', 'file')])},
23631             {'call': 'fchmod', 'reason': set([('fd', 'file')])},
23632             {'call': 'pread64', 'reason': set([('fd', 'file')])},
23633             {'call': 'signalfd4', 'reason': set([('fd', 'file')])},
23634             {'call': 'read', 'reason': set([('fd', 'file')])},
23635             {'call': 'fchown', 'reason': set([('fd', 'file')])},
23636             {'call': 'mq_timedreceive', 'reason': set([('fd', 'file')])},
23637             {'call': 'utime', 'reason': set([('fd', 'file')])},
23638             {'call': 'fsync', 'reason': set([('fd', 'file')])},
23639             {'call': 'bpf', 'reason': set([('fd', 'file')])},
23640             {'call': 'recvfrom', 'reason': set([('fd', 'file')])},
23641             {'call': 'fsetxattr', 'reason': set([('fd', 'file')])},
23642             {'call': 'sendto', 'reason': set([('fd', 'file')])},
23643             {'call': 'tee', 'reason': set([('fd', 'file')])},
23644             {'call': 'sync_file_range', 'reason': set([('fd', 'file')])},
23645             {'call': 'lseek', 'reason': set([('fd', 'file')])},
23646             {'call': 'connect', 'reason': set([('fd', 'file')])},
23647             {'call': 'getsockname', 'reason': set([('fd', 'file')])},
23648             {'call': 'epoll_ctl', 'reason': set([('fd', 'file')])},
23649             {'call': 'flock', 'reason': set([('fd', 'file')])},
23650             {'call': 'pwritev', 'reason': set([('fd', 'file')])},
23651             {'call': 'fchdir', 'reason': set([('fd', 'file')])},
23652             {'call': 'accept4', 'reason': set([('fd', 'file')])},
23653             {'call': 'old_readdir', 'reason': set([('fd', 'file')])},
23654             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'file')])},
23655             {'call': 'utimensat', 'reason': set([('fd', 'file')])},
23656             {'call': 'inotify_add_watch', 'reason': set([('fd', 'file')])},
23657             {'call': 'preadv2', 'reason': set([('fd', 'file')])},
23658             {'call': 'splice', 'reason': set([('fd', 'file')])},
23659             {'call': 'ftruncate', 'reason': set([('fd', 'file')])},
23660             {'call': 'preadv', 'reason': set([('fd', 'file')])},
23661             {'call': 'getpeername', 'reason': set([('fd', 'file')])},
23662             {'call': 'setsockopt', 'reason': set([('fd', 'file')])},
23663             {'call': 'fcntl', 'reason': set([('fd', 'file')])},
23664             {'call': 'ioctl', 'reason': set([('fd', 'file')])},
23665             {'call': 'pwrite64', 'reason': set([('fd', 'file')])},
23666             {'call': 'perf_event_open', 'reason': set([('fd', 'file')])},
23667             {'call': 'pwritev64v2', 'reason': set([('fd', 'file')])},
23668             {'call': 'futimesat', 'reason': set([('fd', 'file')])},
23669             {'call': 'pwritev2', 'reason': set([('fd', 'file')])},
23670             {'call': 'shutdown', 'reason': set([('fd', 'file')])},
23671             {'call': 'getsockopt', 'reason': set([('fd', 'file')])},
23672             {'call': 'mq_getsetattr', 'reason': set([('fd', 'file')])},
23673             {'call': 'fdatasync', 'reason': set([('fd', 'file')])},
23674             {'call': 'getdents64', 'reason': set([('fd', 'file')])},
23675             {'call': 'listen', 'reason': set([('fd', 'file')])},
23676             {'call': 'copy_file_range', 'reason': set([('fd', 'file')])},
23677             {'call': 'mq_timedsend', 'reason': set([('fd', 'file')])},
23678             {'call': 'fgetxattr', 'reason': set([('fd', 'file')])},
23679             {'call': 'fcntl64', 'reason': set([('fd', 'file')])},
23680             {'call': 'fallocate', 'reason': set([('fd', 'file')])},
23681             {'call': 'epoll_wait', 'reason': set([('fd', 'file')])},
23682             {'call': 'llseek', 'reason': set([('fd', 'file')])},
23683             {'call': 'preadv64v2', 'reason': set([('fd', 'file')])},
23684             {'call': 'readv', 'reason': set([('fd', 'file')])},
23685             {'call': 'fstatfs', 'reason': set([('fd', 'file')])},
23686             {'call': 'fstatfs64', 'reason': set([('fd', 'file')])},
23687             {'call': 'write', 'reason': set([('fd', 'file')])},
```



```
23688             {'call': 'mq_notify', 'reason': set([('fd', 'file')])},
23689             {'call': 'sendfile', 'reason': set([('fd', 'file')])},
23690             {'call': 'bind', 'reason': set([('fd', 'file')])},
23691             {'call': 'flistxattr', 'reason': set([('fd', 'file')])},
23692             {'call': 'sendfile64', 'reason': set([('fd', 'file')])}]}
```

```
**********************************************************
 1054698 Fri Dec 21 15:00:33 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/with_structs/i
mplicit_dependencies_verbose.pretty
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
{'acct': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
          {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
          {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
          {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
          {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
          {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
          {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
          {'call': 'flock', 'reason': set([('file', 'f_mode')])},
          {'call': 'openat', 'reason': set([('file', 'f_mode')])},
          {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
          {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
          {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
          {'call': 'socket', 'reason': set([('file', 'f_mode')])},
          {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
          {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
          {'call': 'open', 'reason': set([('file', 'f_mode')])},
          {'call': 'dup', 'reason': set([('file', 'f_mode')])},
          {'call': 'setns', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
          {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
          {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
          {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
          {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])}],
 'alarm': [{'call': 'waitid',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'settimeofday',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'adjtimex',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'getitimer',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'select',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'wait4',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'getrusage',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'setitimer',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'clock_adjtime',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'ppoll',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])}],
 'bpf': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'rt_sigtimedwait',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'eventfd2',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'swapoff',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'sched_getaffinity',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_setparam', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'memfd_create',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
```



```
         {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'remap_file_pages',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'dup3',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'mq_timedreceive',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_setaffinity',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'semtimedop', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_rr_get_interval',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'epoll_create1',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'epoll_ctl',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'flock',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'openat',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'uselib',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'rt_sigprocmask', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'accept4',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_setattr', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'socketpair',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'migrate_pages', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'shmat',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'socket',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'pipe2',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'perf_event_open',
          'reason': set([('bpf_prog', 'aux'),
                         ('file', 'f_op'),
                         ('file', 'private_data'),
                         ('mm_segment_t', 'seg')])},
         {'call': 'shmdt',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'rt_sigaction', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'acct',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'open',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'dup',
          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
         {'call': 'setns',
          'reason': set([('file', 'f_op'),
                         ('file', 'private_data'),
```

```
126                          ('mm_segment_t', 'seg')])},
127         {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
128         {'call': 'get_robust_list',
129          'reason': set([('mm_segment_t', 'seg')])},
130         {'call': 'mq_timedsend', 'reason': set([('mm_segment_t', 'seg')])},
131         {'call': 'sched_getscheduler',
132          'reason': set([('mm_segment_t', 'seg')])},
133         {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
134         {'call': 'shmctl',
135          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
136         {'call': 'swapon',
137          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
138         {'call': 'sched_getattr', 'reason': set([('mm_segment_t', 'seg')])},
139         {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
140         {'call': 'sched_setscheduler',
141          'reason': set([('mm_segment_t', 'seg')])},
142         {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
143         {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
144         {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
145         {'call': 'mmap_pgoff',
146          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
147         {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
148         {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
149         {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
150         {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
151         {'call': 'mq_open',
152          'reason': set([('file', 'f_op'), ('file', 'private_data')])},
153         {'call': 'sched_getparam', 'reason': set([('mm_segment_t', 'seg')])},
154         {'call': 'open_by_handle_at',
155          'reason': set([('file', 'f_op'), ('file', 'private_data')])}],
156  'brk': [{'call': 'swapoff',
157           'reason': set([('mm_struct', 'brk'),
158                          ('mm_struct', 'def_flags'),
159                          ('mm_struct', 'end_data'),
160                          ('mm_struct', 'start_brk'),
161                          ('mm_struct', 'start_data')])},
162         {'call': 'remap_file_pages',
163          'reason': set([('mm_struct', 'brk'),
164                         ('mm_struct', 'def_flags'),
165                         ('mm_struct', 'end_data'),
166                         ('mm_struct', 'start_brk'),
167                         ('mm_struct', 'start_data'),
168                         ('vm_area_struct', 'vm_flags'),
169                         ('vm_area_struct', 'vm_start')])},
170         {'call': 'io_getevents',
171          'reason': set([('mm_struct', 'brk'),
172                         ('mm_struct', 'def_flags'),
173                         ('mm_struct', 'end_data'),
174                         ('mm_struct', 'start_brk'),
175                         ('mm_struct', 'start_data')])},
176         {'call': 'migrate_pages',
177          'reason': set([('mm_struct', 'brk'),
178                         ('mm_struct', 'def_flags'),
179                         ('mm_struct', 'end_data'),
180                         ('mm_struct', 'start_brk'),
181                         ('mm_struct', 'start_data')])},
182         {'call': 'shmdt',
183          'reason': set([('mm_struct', 'brk'),
184                         ('mm_struct', 'def_flags'),
185                         ('mm_struct', 'end_data'),
186                         ('mm_struct', 'start_brk'),
187                         ('mm_struct', 'start_data'),
188                         ('vm_area_struct', 'vm_flags'),
189                         ('vm_area_struct', 'vm_start')])},
190         {'call': 'get_mempolicy',
191          'reason': set([('mm_struct', 'brk'),
```



```
192                         ('mm_struct', 'def_flags'),
193                         ('mm_struct', 'end_data'),
194                         ('mm_struct', 'start_brk'),
195                         ('mm_struct', 'start_data'),
196                         ('vm_area_struct', 'vm_flags'),
197                         ('vm_area_struct', 'vm_start')])},
198         {'call': 'munlockall',
199          'reason': set([('mm_struct', 'def_flags'),
200                         ('vm_area_struct', 'vm_flags'),
201                         ('vm_area_struct', 'vm_start')])},
202         {'call': 'pkey_mprotect',
203          'reason': set([('vm_area_struct', 'vm_flags'),
204                         ('vm_area_struct', 'vm_start')])},
205         {'call': 'madvise',
206          'reason': set([('vm_area_struct', 'vm_flags'),
207                         ('vm_area_struct', 'vm_start')])},
208         {'call': 'getrusage',
209          'reason': set([('mm_struct', 'brk'),
210                         ('mm_struct', 'def_flags'),
211                         ('mm_struct', 'end_data'),
212                         ('mm_struct', 'start_brk'),
213                         ('mm_struct', 'start_data')])},
214         {'call': 'io_setup',
215          'reason': set([('mm_struct', 'brk'),
216                         ('mm_struct', 'def_flags'),
217                         ('mm_struct', 'end_data'),
218                         ('mm_struct', 'start_brk'),
219                         ('mm_struct', 'start_data')])},
220         {'call': 'mprotect',
221          'reason': set([('vm_area_struct', 'vm_flags'),
222                         ('vm_area_struct', 'vm_start')])},
223         {'call': 'mremap',
224          'reason': set([('mm_struct', 'brk'),
225                         ('mm_struct', 'def_flags'),
226                         ('mm_struct', 'end_data'),
227                         ('mm_struct', 'start_brk'),
228                         ('mm_struct', 'start_data'),
229                         ('vm_area_struct', 'vm_flags'),
230                         ('vm_area_struct', 'vm_start')])},
231         {'call': 'io_destroy',
232          'reason': set([('mm_struct', 'brk'),
233                         ('mm_struct', 'def_flags'),
234                         ('mm_struct', 'end_data'),
235                         ('mm_struct', 'start_brk'),
236                         ('mm_struct', 'start_data')])},
237         {'call': 'mbind',
238          'reason': set([('mm_struct', 'brk'),
239                         ('mm_struct', 'def_flags'),
240                         ('mm_struct', 'end_data'),
241                         ('mm_struct', 'start_brk'),
242                         ('mm_struct', 'start_data')])},
243         {'call': 'prctl',
244          'reason': set([('mm_struct', 'brk'),
245                         ('mm_struct', 'def_flags'),
246                         ('mm_struct', 'end_data'),
247                         ('mm_struct', 'start_brk'),
248                         ('mm_struct', 'start_data'),
249                         ('vm_area_struct', 'vm_flags'),
250                         ('vm_area_struct', 'vm_start')])},
251         {'call': 'move_pages',
252          'reason': set([('mm_struct', 'brk'),
253                         ('mm_struct', 'def_flags'),
254                         ('mm_struct', 'end_data'),
255                         ('mm_struct', 'start_brk'),
256                         ('mm_struct', 'start_data')])},
257         {'call': 'modify_ldt',
```

```
258           'reason': set([('mm_struct', 'brk'),
259                          ('mm_struct', 'def_flags'),
260                          ('mm_struct', 'end_data'),
261                          ('mm_struct', 'start_brk'),
262                          ('mm_struct', 'start_data')])},
263          {'call': 'munlock',
264           'reason': set([('vm_area_struct', 'vm_flags'),
265                          ('vm_area_struct', 'vm_start')])},
266          {'call': 'mincore',
267           'reason': set([('mm_struct', 'brk'),
268                          ('mm_struct', 'def_flags'),
269                          ('mm_struct', 'end_data'),
270                          ('mm_struct', 'start_brk'),
271                          ('mm_struct', 'start_data'),
272                          ('vm_area_struct', 'vm_flags'),
273                          ('vm_area_struct', 'vm_start')])},
274          {'call': 'io_cancel',
275           'reason': set([('mm_struct', 'brk'),
276                          ('mm_struct', 'def_flags'),
277                          ('mm_struct', 'end_data'),
278                          ('mm_struct', 'start_brk'),
279                          ('mm_struct', 'start_data')])},
280          {'call': 'mlockall',
281           'reason': set([('mm_struct', 'def_flags'),
282                          ('vm_area_struct', 'vm_flags'),
283                          ('vm_area_struct', 'vm_start')])}],
284  'capset': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
285             {'call': 'rt_sigtimedwait',
286              'reason': set([('task_struct', 'cred')])},
287             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
288             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
289             {'call': 'sched_getaffinity',
290              'reason': set([('task_struct', 'cred')])},
291             {'call': 'sched_setparam',
292              'reason': set([('task_struct', 'cred')])},
293             {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
294             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
295             {'call': 'mq_timedreceive',
296              'reason': set([('task_struct', 'cred')])},
297             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
298             {'call': 'sched_setaffinity',
299              'reason': set([('task_struct', 'cred')])},
300             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
301             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
302             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
303             {'call': 'sched_rr_get_interval',
304              'reason': set([('task_struct', 'cred')])},
305             {'call': 'rt_sigprocmask',
306              'reason': set([('task_struct', 'cred')])},
307             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
308             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
309             {'call': 'sched_setattr',
310              'reason': set([('task_struct', 'cred')])},
311             {'call': 'migrate_pages',
312              'reason': set([('task_struct', 'cred')])},
313             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
314             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
315             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
316             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
317             {'call': 'perf_event_open',
318              'reason': set([('task_struct', 'cred')])},
319             {'call': 'rt_sigaction',
320              'reason': set([('task_struct', 'cred')])},
321             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
322             {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
323             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
```



```
324             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
325             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
326             {'call': 'get_robust_list',
327              'reason': set([('task_struct', 'cred')])},
328             {'call': 'mq_timedsend',
329              'reason': set([('task_struct', 'cred')])},
330             {'call': 'sched_getscheduler',
331              'reason': set([('task_struct', 'cred')])},
332             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
333             {'call': 'sched_getattr',
334              'reason': set([('task_struct', 'cred')])},
335             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
336             {'call': 'sched_setscheduler',
337              'reason': set([('task_struct', 'cred')])},
338             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
339             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
340             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
341             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
342             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
343             {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
344             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
345             {'call': 'sched_getparam',
346              'reason': set([('task_struct', 'cred')])}],
347  'clock_adjtime': [{'call': 'clock_getres',
348                     'reason': set([('k_clock', 'clock_adj')])},
349                    {'call': 'timer_delete',
350                     'reason': set([('k_clock', 'clock_adj')])},
351                    {'call': 'timer_create',
352                     'reason': set([('k_clock', 'clock_adj')])},
353                    {'call': 'clock_gettime',
354                     'reason': set([('k_clock', 'clock_adj')])},
355                    {'call': 'timer_settime',
356                     'reason': set([('k_clock', 'clock_adj')])},
357                    {'call': 'timer_gettime',
358                     'reason': set([('k_clock', 'clock_adj')])},
359                    {'call': 'clock_settime',
360                     'reason': set([('k_clock', 'clock_adj')])},
361                    {'call': 'clock_nanosleep',
362                     'reason': set([('k_clock', 'clock_adj')])}],
363  'clock_nanosleep': [{'call': 'rt_sigtimedwait',
364                       'reason': set([('timespec', 'tv_nsec'),
365                                      ('timespec', 'tv_sec')])},
366                      {'call': 'mq_unlink',
367                       'reason': set([('timespec', 'tv_nsec'),
368                                      ('timespec', 'tv_sec')])},
369                      {'call': 'swapoff',
370                       'reason': set([('timespec', 'tv_nsec'),
371                                      ('timespec', 'tv_sec')])},
372                      {'call': 'clock_getres',
373                       'reason': set([('k_clock', 'nsleep')])},
374                      {'call': 'timer_delete',
375                       'reason': set([('k_clock', 'nsleep')])},
376                      {'call': 'fchmod',
377                       'reason': set([('timespec', 'tv_nsec'),
378                                      ('timespec', 'tv_sec')])},
379                      {'call': 'memfd_create',
380                       'reason': set([('timespec', 'tv_nsec'),
381                                      ('timespec', 'tv_sec')])},
382                      {'call': 'readlinkat',
383                       'reason': set([('timespec', 'tv_nsec'),
384                                      ('timespec', 'tv_sec')])},
385                      {'call': 'io_getevents',
386                       'reason': set([('timespec', 'tv_nsec'),
387                                      ('timespec', 'tv_sec')])},
388                      {'call': 'fchown',
389                       'reason': set([('timespec', 'tv_nsec'),
```

```
390                                 ('timespec', 'tv_sec')])},
391                     {'call': 'mq_timedreceive',
392                      'reason': set([('timespec', 'tv_nsec'),
393                                     ('timespec', 'tv_sec')])},
394                     {'call': 'utime',
395                      'reason': set([('timespec', 'tv_nsec'),
396                                     ('timespec', 'tv_sec')])},
397                     {'call': 'semtimedop',
398                      'reason': set([('timespec', 'tv_nsec'),
399                                     ('timespec', 'tv_sec')])},
400                     {'call': 'settimeofday',
401                      'reason': set([('timespec', 'tv_nsec'),
402                                     ('timespec', 'tv_sec')])},
403                     {'call': 'timer_create',
404                      'reason': set([('k_clock', 'nsleep')])},
405                     {'call': 'clock_gettime',
406                      'reason': set([('k_clock', 'nsleep')])},
407                     {'call': 'sched_rr_get_interval',
408                      'reason': set([('timespec', 'tv_nsec'),
409                                     ('timespec', 'tv_sec')])},
410                     {'call': 'timerfd_gettime',
411                      'reason': set([('timespec', 'tv_nsec'),
412                                     ('timespec', 'tv_sec')])},
413                     {'call': 'pselect6',
414                      'reason': set([('timespec', 'tv_nsec'),
415                                     ('timespec', 'tv_sec')])},
416                     {'call': 'uselib',
417                      'reason': set([('timespec', 'tv_nsec'),
418                                     ('timespec', 'tv_sec')])},
419                     {'call': 'fchmodat',
420                      'reason': set([('timespec', 'tv_nsec'),
421                                     ('timespec', 'tv_sec')])},
422                     {'call': 'inotify_add_watch',
423                      'reason': set([('timespec', 'tv_nsec'),
424                                     ('timespec', 'tv_sec')])},
425                     {'call': 'timer_settime',
426                      'reason': set([('k_clock', 'nsleep'),
427                                     ('timespec', 'tv_nsec'),
428                                     ('timespec', 'tv_sec')])},
429                     {'call': 'ftruncate',
430                      'reason': set([('timespec', 'tv_nsec'),
431                                     ('timespec', 'tv_sec')])},
432                     {'call': 'timer_gettime',
433                      'reason': set([('k_clock', 'nsleep'),
434                                     ('timespec', 'tv_nsec'),
435                                     ('timespec', 'tv_sec')])},
436                     {'call': 'ioctl',
437                      'reason': set([('timespec', 'tv_nsec'),
438                                     ('timespec', 'tv_sec')])},
439                     {'call': 'linkat',
440                      'reason': set([('timespec', 'tv_nsec'),
441                                     ('timespec', 'tv_sec')])},
442                     {'call': 'stime',
443                      'reason': set([('timespec', 'tv_nsec'),
444                                     ('timespec', 'tv_sec')])},
445                     {'call': 'futimesat',
446                      'reason': set([('timespec', 'tv_nsec'),
447                                     ('timespec', 'tv_sec')])},
448                     {'call': 'poll',
449                      'reason': set([('timespec', 'tv_nsec'),
450                                     ('timespec', 'tv_sec')])},
451                     {'call': 'clock_settime',
452                      'reason': set([('k_clock', 'nsleep')])},
453                     {'call': 'select',
454                      'reason': set([('timespec', 'tv_nsec'),
455                                     ('timespec', 'tv_sec')])},
```



```
456                     {'call': 'unlink',
457                      'reason': set([('timespec', 'tv_nsec'),
458                                     ('timespec', 'tv_sec')])},
459                     {'call': 'nanosleep',
460                      'reason': set([('timespec', 'tv_nsec'),
461                                     ('timespec', 'tv_sec')])},
462                     {'call': 'mq_getsetattr',
463                      'reason': set([('timespec', 'tv_nsec'),
464                                     ('timespec', 'tv_sec')])},
465                     {'call': 'faccessat',
466                      'reason': set([('timespec', 'tv_nsec'),
467                                     ('timespec', 'tv_sec')])},
468                     {'call': 'mq_timedsend',
469                      'reason': set([('timespec', 'tv_nsec'),
470                                     ('timespec', 'tv_sec')])},
471                     {'call': 'swapon',
472                      'reason': set([('timespec', 'tv_nsec'),
473                                     ('timespec', 'tv_sec')])},
474                     {'call': 'epoll_wait',
475                      'reason': set([('timespec', 'tv_nsec'),
476                                     ('timespec', 'tv_sec')])},
477                     {'call': 'fchownat',
478                      'reason': set([('timespec', 'tv_nsec'),
479                                     ('timespec', 'tv_sec')])},
480                     {'call': 'fstat',
481                      'reason': set([('timespec', 'tv_nsec'),
482                                     ('timespec', 'tv_sec')])},
483                     {'call': 'timerfd_settime',
484                      'reason': set([('timespec', 'tv_nsec'),
485                                     ('timespec', 'tv_sec')])},
486                     {'call': 'mq_notify',
487                      'reason': set([('timespec', 'tv_nsec'),
488                                     ('timespec', 'tv_sec')])},
489                     {'call': 'sendfile',
490                      'reason': set([('timespec', 'tv_nsec'),
491                                     ('timespec', 'tv_sec')])},
492                     {'call': 'newfstat',
493                      'reason': set([('timespec', 'tv_nsec'),
494                                     ('timespec', 'tv_sec')])},
495                     {'call': 'unlinkat',
496                      'reason': set([('timespec', 'tv_nsec'),
497                                     ('timespec', 'tv_sec')])},
498                     {'call': 'clock_adjtime',
499                      'reason': set([('k_clock', 'nsleep')])},
500                     {'call': 'futex',
501                      'reason': set([('timespec', 'tv_nsec'),
502                                     ('timespec', 'tv_sec')])},
503                     {'call': 'recvmmsg',
504                      'reason': set([('timespec', 'tv_nsec'),
505                                     ('timespec', 'tv_sec')])},
506                     {'call': 'sendfile64',
507                      'reason': set([('timespec', 'tv_nsec'),
508                                     ('timespec', 'tv_sec')])},
509                     {'call': 'ppoll',
510                      'reason': set([('timespec', 'tv_nsec'),
511                                     ('timespec', 'tv_sec')])}],
512  'clock_settime': [{'call': 'clock_getres',
513                     'reason': set([('k_clock', 'clock_set')])},
514                    {'call': 'timer_delete',
515                     'reason': set([('k_clock', 'clock_set')])},
516                    {'call': 'timer_create',
517                     'reason': set([('k_clock', 'clock_set')])},
518                    {'call': 'clock_gettime',
519                     'reason': set([('k_clock', 'clock_set')])},
520                    {'call': 'timer_settime',
521                     'reason': set([('k_clock', 'clock_set')])},
```

```
                  {'call': 'timer_gettime',
                   'reason': set([('k_clock', 'clock_set')])},
                  {'call': 'clock_nanosleep',
                   'reason': set([('k_clock', 'clock_set')])},
                  {'call': 'clock_adjtime',
                   'reason': set([('k_clock', 'clock_set')])}],
 'copy_file_range': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                     {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                     {'call': 'fremovexattr',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'readahead', 'reason': set([('fd', 'flags')])},
                     {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                     {'call': 'writev', 'reason': set([('fd', 'flags')])},
                     {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                     {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                     {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                     {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                     {'call': 'read', 'reason': set([('fd', 'flags')])},
                     {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                     {'call': 'mq_timedreceive',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'utime', 'reason': set([('fd', 'flags')])},
                     {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                     {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                     {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                     {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                     {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                     {'call': 'tee', 'reason': set([('fd', 'flags')])},
                     {'call': 'sync_file_range',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                     {'call': 'connect', 'reason': set([('fd', 'flags')])},
                     {'call': 'getsockname',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
                     {'call': 'flock', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                     {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                     {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                     {'call': 'old_readdir',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'inotify_rm_watch',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                     {'call': 'inotify_add_watch',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                     {'call': 'splice', 'reason': set([('fd', 'flags')])},
                     {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                     {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                     {'call': 'getpeername',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                     {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                     {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                     {'call': 'perf_event_open',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev64v2',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                     {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                     {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                     {'call': 'mq_getsetattr',
```



```
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                     {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                     {'call': 'listen', 'reason': set([('fd', 'flags')])},
                     {'call': 'mq_timedsend',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                     {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                     {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                     {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                     {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                     {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                     {'call': 'readv', 'reason': set([('fd', 'flags')])},
                     {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                     {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                     {'call': 'write', 'reason': set([('fd', 'flags')])},
                     {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                     {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                     {'call': 'bind', 'reason': set([('fd', 'flags')])},
                     {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                     {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'delete_module': [{'call': 'init_module',
                    'reason': set([('module', 'exit'),
                                   ('module', 'init'),
                                   ('module', 'state')])},
                   {'call': 'finit_module',
                    'reason': set([('module', 'exit'),
                                   ('module', 'init'),
                                   ('module', 'state')])}],
 'dup3': [{'call': 'unshare',
           'reason': set([('fdtable', 'max_fds'),
                          ('files_struct', 'fdt'),
                          ('files_struct', 'resize_in_progress')])},
          {'call': 'select', 'reason': set([('fdtable', 'max_fds')])},
          {'call': 'dup2',
           'reason': set([('fdtable', 'max_fds'),
                          ('files_struct', 'fdt'),
                          ('files_struct', 'resize_in_progress')])}],
 'epoll_create1': [{'call': 'keyctl',
                    'reason': set([('cred', 'user'),
                                   ('task_struct', 'cred')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setfsuid', 'reason': set([('cred', 'user')])},
                   {'call': 'msgrcv',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
                   {'call': 'getresuid16', 'reason': set([('cred', 'user')])},
                   {'call': 'getresgid', 'reason': set([('cred', 'user')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setparam',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setgid', 'reason': set([('cred', 'user')])},
                   {'call': 'ioprio_set',
                    'reason': set([('cred', 'user'),
                                   ('task_struct', 'cred')])},
                   {'call': 'capset', 'reason': set([('cred', 'user')])},
                   {'call': 'getppid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getresgid16', 'reason': set([('cred', 'user')])},
                   {'call': 'capget',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setaffinity',
```

```
654                      'reason': set([('cred', 'user'),
655                                     ('task_struct', 'cred')])},
656                     {'call': 'setfsgid', 'reason': set([('cred', 'user')])},
657                     {'call': 'unshare', 'reason': set([('cred', 'user')])},
658                     {'call': 'signal',
659                      'reason': set([('task_struct', 'cred')])},
660                     {'call': 'setreuid', 'reason': set([('cred', 'user')])},
661                     {'call': 'semtimedop',
662                      'reason': set([('task_struct', 'cred')])},
663                     {'call': 'umount',
664                      'reason': set([('task_struct', 'cred')])},
665                     {'call': 'sched_rr_get_interval',
666                      'reason': set([('task_struct', 'cred')])},
667                     {'call': 'getresuid', 'reason': set([('cred', 'user')])},
668                     {'call': 'epoll_ctl',
669                      'reason': set([('eventpoll', 'user'),
670                                     ('eventpoll', 'ws')])},
671                     {'call': 'rt_sigprocmask',
672                      'reason': set([('task_struct', 'cred')])},
673                     {'call': 'setsid',
674                      'reason': set([('task_struct', 'cred')])},
675                     {'call': 'sigaltstack',
676                      'reason': set([('task_struct', 'cred')])},
677                     {'call': 'sched_setattr',
678                      'reason': set([('task_struct', 'cred')])},
679                     {'call': 'migrate_pages',
680                      'reason': set([('cred', 'user'),
681                                     ('task_struct', 'cred')])},
682                     {'call': 'getitimer',
683                      'reason': set([('task_struct', 'cred')])},
684                     {'call': 'setpgid',
685                      'reason': set([('task_struct', 'cred')])},
686                     {'call': 'setresgid', 'reason': set([('cred', 'user')])},
687                     {'call': 'setregid', 'reason': set([('cred', 'user')])},
688                     {'call': 'getsid',
689                      'reason': set([('task_struct', 'cred')])},
690                     {'call': 'prlimit64',
691                      'reason': set([('cred', 'user'),
692                                     ('task_struct', 'cred')])},
693                     {'call': 'perf_event_open',
694                      'reason': set([('task_struct', 'cred')])},
695                     {'call': 'getgroups16', 'reason': set([('cred', 'user')])},
696                     {'call': 'rt_sigaction',
697                      'reason': set([('task_struct', 'cred')])},
698                     {'call': 'getpgid',
699                      'reason': set([('task_struct', 'cred')])},
700                     {'call': 'getpriority',
701                      'reason': set([('cred', 'user'),
702                                     ('task_struct', 'cred')])},
703                     {'call': 'sigaction',
704                      'reason': set([('task_struct', 'cred')])},
705                     {'call': 'faccessat', 'reason': set([('cred', 'user')])},
706                     {'call': 'setns',
707                      'reason': set([('task_struct', 'cred')])},
708                     {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
709                     {'call': 'get_robust_list',
710                      'reason': set([('task_struct', 'cred')])},
711                     {'call': 'mq_timedsend',
712                      'reason': set([('task_struct', 'cred')])},
713                     {'call': 'sched_getscheduler',
714                      'reason': set([('task_struct', 'cred')])},
715                     {'call': 'ptrace',
716                      'reason': set([('task_struct', 'cred')])},
717                     {'call': 'epoll_wait',
718                      'reason': set([('eventpoll', 'user'),
719                                     ('eventpoll', 'ws')])},
```



```
720                     {'call': 'sched_getattr',
721                      'reason': set([('task_struct', 'cred')])},
722                     {'call': 'getrusage',
723                      'reason': set([('task_struct', 'cred')])},
724                     {'call': 'sched_setscheduler',
725                      'reason': set([('task_struct', 'cred')])},
726                     {'call': 'setresuid', 'reason': set([('cred', 'user')])},
727                     {'call': 'setitimer',
728                      'reason': set([('task_struct', 'cred')])},
729                     {'call': 'ioprio_get',
730                      'reason': set([('cred', 'user'),
731                                     ('task_struct', 'cred')])},
732                     {'call': 'vfork',
733                      'reason': set([('task_struct', 'cred')])},
734                     {'call': 'setuid', 'reason': set([('cred', 'user')])},
735                     {'call': 'prctl',
736                      'reason': set([('task_struct', 'cred')])},
737                     {'call': 'move_pages',
738                      'reason': set([('task_struct', 'cred')])},
739                     {'call': 'getgroups', 'reason': set([('cred', 'user')])},
740                     {'call': 'setpriority',
741                      'reason': set([('cred', 'user'),
742                                     ('task_struct', 'cred')])},
743                     {'call': 'clone',
744                      'reason': set([('task_struct', 'cred')])},
745                     {'call': 'sched_getparam',
746                      'reason': set([('task_struct', 'cred')])}],
747  'epoll_ctl': [{'call': 'syncfs',
748                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
749                {'call': 'keyctl', 'reason': set([('list_head', 'next')])},
750                {'call': 'rt_sigtimedwait',
751                 'reason': set([('list_head', 'next')])},
752                {'call': 'vmsplice',
753                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
754                {'call': 'msgrcv', 'reason': set([('list_head', 'next')])},
755                {'call': 'eventfd2',
756                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
757                {'call': 'mq_unlink', 'reason': set([('list_head', 'next')])},
758                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
759                {'call': 'kill', 'reason': set([('list_head', 'next')])},
760                {'call': 'swapoff',
761                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
762                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
763                {'call': 'readahead',
764                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
765                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
766                {'call': 'timer_delete',
767                 'reason': set([('list_head', 'next')])},
768                {'call': 'sched_getaffinity',
769                 'reason': set([('list_head', 'next')])},
770                {'call': 'writev', 'reason': set([('fd', 'flags')])},
771                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
772                {'call': 'sched_setparam',
773                 'reason': set([('list_head', 'next')])},
774                {'call': 'fchmod',
775                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
776                {'call': 'setgid', 'reason': set([('list_head', 'next')])},
777                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
778                {'call': 'pivot_root', 'reason': set([('list_head', 'next')])},
779                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
780                {'call': 'memfd_create',
781                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
782                {'call': 'ioprio_set', 'reason': set([('list_head', 'next')])},
783                {'call': 'delete_module',
784                 'reason': set([('list_head', 'next')])},
785                {'call': 'remap_file_pages',
```

```
786                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
787                {'call': 'dup3',
788                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
789                {'call': 'readlinkat', 'reason': set([('list_head', 'next')])},
790                {'call': 'read', 'reason': set([('fd', 'flags')])},
791                {'call': 'io_getevents',
792                 'reason': set([('list_head', 'next')])},
793                {'call': 'getppid', 'reason': set([('list_head', 'next')])},
794                {'call': 'fchown',
795                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
796                {'call': 'mq_timedreceive',
797                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
798                {'call': 'capget', 'reason': set([('list_head', 'next')])},
799                {'call': 'utime', 'reason': set([('fd', 'flags')])},
800                {'call': 'sched_setaffinity',
801                 'reason': set([('list_head', 'next')])},
802                {'call': 'ustat', 'reason': set([('list_head', 'next')])},
803                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
804                {'call': 'bpf',
805                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
806                {'call': 'unshare', 'reason': set([('list_head', 'next')])},
807                {'call': 'signal', 'reason': set([('list_head', 'next')])},
808                {'call': 'setreuid', 'reason': set([('list_head', 'next')])},
809                {'call': 'semtimedop', 'reason': set([('list_head', 'next')])},
810                {'call': 'umount', 'reason': set([('list_head', 'next')])},
811                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
812                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
813                {'call': 'timer_create',
814                 'reason': set([('list_head', 'next')])},
815                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
816                {'call': 'mkdirat', 'reason': set([('list_head', 'next')])},
817                {'call': 'sched_rr_get_interval',
818                 'reason': set([('list_head', 'next')])},
819                {'call': 'epoll_create1',
820                 'reason': set([('epitem', 'nwait'),
821                                ('epitem', 'ws'),
822                                ('epoll_event', 'events'),
823                                ('file', 'f_op'),
824                                ('list_head', 'next')])},
825                {'call': 'timerfd_gettime',
826                 'reason': set([('list_head', 'next')])},
827                {'call': 'tee',
828                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
829                {'call': 'semctl', 'reason': set([('list_head', 'next')])},
830                {'call': 'sync_file_range',
831                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
832                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
833                {'call': 'connect', 'reason': set([('fd', 'flags')])},
834                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
835                {'call': 'flock',
836                 'reason': set([('fd', 'flags'),
837                                ('file', 'f_op'),
838                                ('list_head', 'next')])},
839                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
840                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
841                {'call': 'openat',
842                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
843                {'call': 'lookup_dcookie',
844                 'reason': set([('list_head', 'next')])},
845                {'call': 'uselib',
846                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
847                {'call': 'renameat2', 'reason': set([('list_head', 'next')])},
848                {'call': 'rt_sigprocmask',
849                 'reason': set([('list_head', 'next')])},
850                {'call': 'accept4',
851                 'reason': set([('fd', 'flags'),
```



```
852                                ('file', 'f_op'),
853                                ('list_head', 'next')])},
854                {'call': 'msgctl', 'reason': set([('list_head', 'next')])},
855                {'call': 'reboot', 'reason': set([('list_head', 'next')])},
856                {'call': 'setsid', 'reason': set([('list_head', 'next')])},
857                {'call': 'set_trip_temp',
858                 'reason': set([('list_head', 'next')])},
859                {'call': 'sigaltstack',
860                 'reason': set([('list_head', 'next')])},
861                {'call': 'sched_setattr',
862                 'reason': set([('list_head', 'next')])},
863                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
864                {'call': 'inotify_rm_watch',
865                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
866                {'call': 'socketpair',
867                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
868                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
869                {'call': 'migrate_pages',
870                 'reason': set([('list_head', 'next')])},
871                {'call': 'getitimer', 'reason': set([('list_head', 'next')])},
872                {'call': 'fchmodat', 'reason': set([('list_head', 'next')])},
873                {'call': 'setpgid', 'reason': set([('list_head', 'next')])},
874                {'call': 'init_module',
875                 'reason': set([('list_head', 'next')])},
876                {'call': 'setresgid', 'reason': set([('list_head', 'next')])},
877                {'call': 'getcwd', 'reason': set([('list_head', 'next')])},
878                {'call': 'inotify_add_watch',
879                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
880                {'call': 'get_trip_temp',
881                 'reason': set([('list_head', 'next')])},
882                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
883                {'call': 'timer_settime',
884                 'reason': set([('list_head', 'next')])},
885                {'call': 'setregid', 'reason': set([('list_head', 'next')])},
886                {'call': 'splice',
887                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
888                {'call': 'ftruncate',
889                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
890                {'call': 'timer_gettime',
891                 'reason': set([('list_head', 'next')])},
892                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
893                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
894                {'call': 'getsid', 'reason': set([('list_head', 'next')])},
895                {'call': 'shmat',
896                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
897                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
898                {'call': 'mknodat', 'reason': set([('list_head', 'next')])},
899                {'call': 'socket',
900                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
901                {'call': 'symlinkat', 'reason': set([('list_head', 'next')])},
902                {'call': 'pipe2',
903                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
904                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
905                {'call': 'ioctl',
906                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
907                {'call': 'prlimit64', 'reason': set([('list_head', 'next')])},
908                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
909                {'call': 'perf_event_open',
910                 'reason': set([('fd', 'flags'),
911                                ('file', 'f_op'),
912                                ('list_head', 'next')])},
913                {'call': 'linkat', 'reason': set([('list_head', 'next')])},
914                {'call': 'getgroups16',
915                 'reason': set([('list_head', 'next')])},
916                {'call': 'shmdt',
917                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
```

```
 918                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
 919                {'call': 'quotactl', 'reason': set([('list_head', 'next')])},
 920                {'call': 'rt_sigaction',
 921                 'reason': set([('list_head', 'next')])},
 922                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
 923                {'call': 'request_key',
 924                 'reason': set([('list_head', 'next')])},
 925                {'call': 'getpgid', 'reason': set([('list_head', 'next')])},
 926                {'call': 'brk', 'reason': set([('list_head', 'next')])},
 927                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
 928                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
 929                {'call': 'acct',
 930                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
 931                {'call': 'open',
 932                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
 933                {'call': 'unlink', 'reason': set([('list_head', 'next')])},
 934                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
 935                {'call': 'exit_group', 'reason': set([('list_head', 'next')])},
 936                {'call': 'getpriority',
 937                 'reason': set([('list_head', 'next')])},
 938                {'call': 'sigaction', 'reason': set([('list_head', 'next')])},
 939                {'call': 'mq_getsetattr',
 940                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
 941                {'call': 'faccessat', 'reason': set([('list_head', 'next')])},
 942                {'call': 'rmdir', 'reason': set([('list_head', 'next')])},
 943                {'call': 'dup',
 944                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
 945                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
 946                {'call': 'setgroups16',
 947                 'reason': set([('list_head', 'next')])},
 948                {'call': 'setns',
 949                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
 950                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
 951                {'call': 'listen', 'reason': set([('fd', 'flags')])},
 952                {'call': 'fork', 'reason': set([('list_head', 'next')])},
 953                {'call': 'get_mempolicy',
 954                 'reason': set([('list_head', 'next')])},
 955                {'call': 'io_submit', 'reason': set([('list_head', 'next')])},
 956                {'call': 'get_robust_list',
 957                 'reason': set([('list_head', 'next')])},
 958                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
 959                {'call': 'mq_timedsend',
 960                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
 961                {'call': 'sched_yield',
 962                 'reason': set([('list_head', 'next')])},
 963                {'call': 'sched_getscheduler',
 964                 'reason': set([('list_head', 'next')])},
 965                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
 966                {'call': 'ptrace', 'reason': set([('list_head', 'next')])},
 967                {'call': 'shmctl',
 968                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
 969                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
 970                {'call': 'munlockall', 'reason': set([('list_head', 'next')])},
 971                {'call': 'swapon',
 972                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
 973                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
 974                {'call': 'pkey_mprotect',
 975                 'reason': set([('list_head', 'next')])},
 976                {'call': 'madvise', 'reason': set([('list_head', 'next')])},
 977                {'call': 'epoll_wait',
 978                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
 979                {'call': 'sched_getattr',
 980                 'reason': set([('list_head', 'next')])},
 981                {'call': 'fchownat', 'reason': set([('list_head', 'next')])},
 982                {'call': 'getrusage', 'reason': set([('list_head', 'next')])},
 983                {'call': 'timerfd_settime',
```



```
 984                 'reason': set([('list_head', 'next')])},
 985                {'call': 'sched_setscheduler',
 986                 'reason': set([('list_head', 'next')])},
 987                {'call': 'setresuid', 'reason': set([('list_head', 'next')])},
 988                {'call': 'setitimer', 'reason': set([('list_head', 'next')])},
 989                {'call': 'ioprio_get', 'reason': set([('list_head', 'next')])},
 990                {'call': 'vfork', 'reason': set([('list_head', 'next')])},
 991                {'call': 'setuid', 'reason': set([('list_head', 'next')])},
 992                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
 993                {'call': 'io_setup', 'reason': set([('list_head', 'next')])},
 994                {'call': 'mprotect', 'reason': set([('list_head', 'next')])},
 995                {'call': 'mmap_pgoff',
 996                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
 997                {'call': 'mremap', 'reason': set([('list_head', 'next')])},
 998                {'call': 'io_destroy', 'reason': set([('list_head', 'next')])},
 999                {'call': 'mbind', 'reason': set([('list_head', 'next')])},
1000                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
1001                {'call': 'readv', 'reason': set([('fd', 'flags')])},
1002                {'call': 'prctl', 'reason': set([('list_head', 'next')])},
1003                {'call': 'move_pages', 'reason': set([('list_head', 'next')])},
1004                {'call': 'timerfd_create',
1005                 'reason': set([('list_head', 'next')])},
1006                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
1007                {'call': 'modify_ldt', 'reason': set([('list_head', 'next')])},
1008                {'call': 'getgroups', 'reason': set([('list_head', 'next')])},
1009                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
1010                {'call': 'dup2', 'reason': set([('list_head', 'next')])},
1011                {'call': 'get_curr_temp',
1012                 'reason': set([('list_head', 'next')])},
1013                {'call': 'msgsnd', 'reason': set([('list_head', 'next')])},
1014                {'call': 'write', 'reason': set([('fd', 'flags')])},
1015                {'call': 'munlock', 'reason': set([('list_head', 'next')])},
1016                {'call': 'setpriority',
1017                 'reason': set([('list_head', 'next')])},
1018                {'call': 'inotify_init1',
1019                 'reason': set([('list_head', 'next')])},
1020                {'call': 'mincore', 'reason': set([('list_head', 'next')])},
1021                {'call': 'mq_notify',
1022                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
1023                {'call': 'sendfile',
1024                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
1025                {'call': 'timer_getoverrun',
1026                 'reason': set([('list_head', 'next')])},
1027                {'call': 'kexec_load', 'reason': set([('list_head', 'next')])},
1028                {'call': 'clone', 'reason': set([('list_head', 'next')])},
1029                {'call': 'mq_open',
1030                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1031                {'call': 'setgroups', 'reason': set([('list_head', 'next')])},
1032                {'call': 'unlinkat', 'reason': set([('list_head', 'next')])},
1033                {'call': 'sched_getparam',
1034                 'reason': set([('list_head', 'next')])},
1035                {'call': 'io_cancel', 'reason': set([('list_head', 'next')])},
1036                {'call': 'open_by_handle_at',
1037                 'reason': set([('file', 'f_op'), ('list_head', 'next')])},
1038                {'call': 'bind', 'reason': set([('fd', 'flags')])},
1039                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
1040                {'call': 'finit_module',
1041                 'reason': set([('list_head', 'next')])},
1042                {'call': 'sendfile64',
1043                 'reason': set([('fd', 'flags'), ('list_head', 'next')])},
1044                {'call': 'mlockall', 'reason': set([('list_head', 'next')])}],
1045  'epoll_wait': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
1046                 {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
1047                 {'call': 'rt_sigtimedwait',
1048                  'reason': set([('mm_segment_t', 'seg')])},
1049                 {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
```

```
1050                {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
1051                {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
1052                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
1053                {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
1054                {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
1055                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
1056                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
1057                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
1058                {'call': 'sched_getaffinity',
1059                 'reason': set([('mm_segment_t', 'seg')])},
1060                {'call': 'writev', 'reason': set([('fd', 'flags')])},
1061                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
1062                {'call': 'sched_setparam',
1063                 'reason': set([('mm_segment_t', 'seg')])},
1064                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
1065                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
1066                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
1067                {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
1068                {'call': 'ioprio_set',
1069                 'reason': set([('mm_segment_t', 'seg')])},
1070                {'call': 'remap_file_pages',
1071                 'reason': set([('file', 'f_op')])},
1072                {'call': 'dup3', 'reason': set([('file', 'f_op')])},
1073                {'call': 'read', 'reason': set([('fd', 'flags')])},
1074                {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
1075                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
1076                {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
1077                {'call': 'mq_timedreceive',
1078                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
1079                {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
1080                {'call': 'utime', 'reason': set([('fd', 'flags')])},
1081                {'call': 'sched_setaffinity',
1082                 'reason': set([('mm_segment_t', 'seg')])},
1083                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
1084                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
1085                {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
1086                {'call': 'semtimedop',
1087                 'reason': set([('mm_segment_t', 'seg')])},
1088                {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
1089                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
1090                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
1091                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
1092                {'call': 'sched_rr_get_interval',
1093                 'reason': set([('mm_segment_t', 'seg')])},
1094                {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
1095                {'call': 'tee', 'reason': set([('fd', 'flags')])},
1096                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
1097                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
1098                {'call': 'connect', 'reason': set([('fd', 'flags')])},
1099                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
1100                {'call': 'epoll_ctl',
1101                 'reason': set([('fd', 'flags'), ('file', 'f_op')])},
1102                {'call': 'flock',
1103                 'reason': set([('fd', 'flags'), ('file', 'f_op')])},
1104                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
1105                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
1106                {'call': 'openat', 'reason': set([('file', 'f_op')])},
1107                {'call': 'uselib', 'reason': set([('file', 'f_op')])},
1108                {'call': 'rt_sigprocmask',
1109                 'reason': set([('mm_segment_t', 'seg')])},
1110                {'call': 'accept4',
1111                 'reason': set([('fd', 'flags'), ('file', 'f_op')])},
1112                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
1113                {'call': 'sigaltstack',
1114                 'reason': set([('mm_segment_t', 'seg')])},
1115                {'call': 'sched_setattr',
```



```
1116                 'reason': set([('mm_segment_t', 'seg')])},
1117                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
1118                {'call': 'inotify_rm_watch',
1119                 'reason': set([('fd', 'flags')])},
1120                {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
1121                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
1122                {'call': 'migrate_pages',
1123                 'reason': set([('mm_segment_t', 'seg')])},
1124                {'call': 'getitimer',
1125                 'reason': set([('mm_segment_t', 'seg')])},
1126                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
1127                {'call': 'inotify_add_watch',
1128                 'reason': set([('fd', 'flags')])},
1129                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
1130                {'call': 'splice', 'reason': set([('fd', 'flags')])},
1131                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
1132                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
1133                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
1134                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
1135                {'call': 'shmat', 'reason': set([('file', 'f_op')])},
1136                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
1137                {'call': 'socket', 'reason': set([('file', 'f_op')])},
1138                {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
1139                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
1140                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
1141                {'call': 'prlimit64',
1142                 'reason': set([('mm_segment_t', 'seg')])},
1143                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
1144                {'call': 'perf_event_open',
1145                 'reason': set([('fd', 'flags'),
1146                                ('file', 'f_op'),
1147                                ('mm_segment_t', 'seg')])},
1148                {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
1149                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
1150                {'call': 'rt_sigaction',
1151                 'reason': set([('mm_segment_t', 'seg')])},
1152                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
1153                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
1154                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
1155                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
1156                {'call': 'acct', 'reason': set([('file', 'f_op')])},
1157                {'call': 'open', 'reason': set([('file', 'f_op')])},
1158                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
1159                {'call': 'getpriority',
1160                 'reason': set([('mm_segment_t', 'seg')])},
1161                {'call': 'sigaction',
1162                 'reason': set([('mm_segment_t', 'seg')])},
1163                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
1164                {'call': 'dup', 'reason': set([('file', 'f_op')])},
1165                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
1166                {'call': 'setns',
1167                 'reason': set([('file', 'f_op'), ('mm_segment_t', 'seg')])},
1168                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
1169                {'call': 'listen', 'reason': set([('fd', 'flags')])},
1170                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
1171                {'call': 'get_robust_list',
1172                 'reason': set([('mm_segment_t', 'seg')])},
1173                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
1174                {'call': 'mq_timedsend',
1175                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
1176                {'call': 'sched_getscheduler',
1177                 'reason': set([('mm_segment_t', 'seg')])},
1178                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
1179                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
1180                {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
1181                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
```

```
                {'call': 'swapon', 'reason': set([('file', 'f_op')])},
                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                {'call': 'sched_getattr',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'getrusage',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'setitimer',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'ioprio_get',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'readv', 'reason': set([('fd', 'flags')])},
                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'move_pages',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                {'call': 'write', 'reason': set([('fd', 'flags')])},
                {'call': 'setpriority',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
                {'call': 'sched_getparam',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_op')])},
                {'call': 'bind', 'reason': set([('fd', 'flags')])},
                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'faccessat': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
               {'call': 'eventfd2', 'reason': set([('path', 'mnt')])},
               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
               {'call': 'swapoff', 'reason': set([('path', 'mnt')])},
               {'call': 'sched_getaffinity',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setparam',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'pivot_root', 'reason': set([('path', 'mnt')])},
               {'call': 'memfd_create', 'reason': set([('path', 'mnt')])},
               {'call': 'ioprio_set',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'remap_file_pages', 'reason': set([('path', 'mnt')])},
               {'call': 'dup3', 'reason': set([('path', 'mnt')])},
               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedreceive',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setaffinity',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'unshare', 'reason': set([('path', 'mnt')])},
               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
               {'call': 'setreuid', 'reason': set([('cred', 'uid')])},
               {'call': 'semtimedop',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_rr_get_interval',
```



```
                'reason': set([('task_struct', 'cred')])},
               {'call': 'epoll_create1', 'reason': set([('path', 'mnt')])},
               {'call': 'epoll_ctl', 'reason': set([('path', 'mnt')])},
               {'call': 'flock', 'reason': set([('path', 'mnt')])},
               {'call': 'openat', 'reason': set([('path', 'mnt')])},
               {'call': 'lookup_dcookie', 'reason': set([('path', 'mnt')])},
               {'call': 'uselib', 'reason': set([('path', 'mnt')])},
               {'call': 'rt_sigprocmask',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'accept4', 'reason': set([('path', 'mnt')])},
               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaltstack',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'socketpair', 'reason': set([('path', 'mnt')])},
               {'call': 'migrate_pages',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'getcwd', 'reason': set([('path', 'mnt')])},
               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'shmat', 'reason': set([('path', 'mnt')])},
               {'call': 'socket', 'reason': set([('path', 'mnt')])},
               {'call': 'pipe2', 'reason': set([('path', 'mnt')])},
               {'call': 'prlimit64',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'perf_event_open',
                'reason': set([('path', 'mnt'), ('task_struct', 'cred')])},
               {'call': 'shmdt', 'reason': set([('path', 'mnt')])},
               {'call': 'quotactl', 'reason': set([('path', 'mnt')])},
               {'call': 'rt_sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'acct', 'reason': set([('path', 'mnt')])},
               {'call': 'open', 'reason': set([('path', 'mnt')])},
               {'call': 'getpriority',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'dup', 'reason': set([('path', 'mnt')])},
               {'call': 'setns',
                'reason': set([('path', 'mnt'), ('task_struct', 'cred')])},
               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
               {'call': 'get_robust_list',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedsend',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getscheduler',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
               {'call': 'shmctl', 'reason': set([('path', 'mnt')])},
               {'call': 'swapon', 'reason': set([('path', 'mnt')])},
               {'call': 'sched_getattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getrusage',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setscheduler',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setresuid', 'reason': set([('cred', 'uid')])},
               {'call': 'setitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ioprio_get',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
```

```
                {'call': 'setuid', 'reason': set([('cred', 'uid')])},
                {'call': 'mmap_pgoff', 'reason': set([('path', 'mnt')])},
                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_open', 'reason': set([('path', 'mnt')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('path', 'mnt')])}],
 'fallocate': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                {'call': 'writev', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                {'call': 'read', 'reason': set([('fd', 'flags')])},
                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
                {'call': 'utime', 'reason': set([('fd', 'flags')])},
                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                {'call': 'tee', 'reason': set([('fd', 'flags')])},
                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                {'call': 'connect', 'reason': set([('fd', 'flags')])},
                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
                {'call': 'flock', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('fd', 'flags')])},
                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                {'call': 'splice', 'reason': set([('fd', 'flags')])},
                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
```



```
                {'call': 'listen', 'reason': set([('fd', 'flags')])},
                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'readv', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                {'call': 'write', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                {'call': 'bind', 'reason': set([('fd', 'flags')])},
                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fchdir': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
            {'call': 'remap_file_pages', 'reason': set([('path', 'dentry')])},
            {'call': 'dup3', 'reason': set([('path', 'dentry')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'unshare', 'reason': set([('path', 'dentry')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'flock',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'openat', 'reason': set([('path', 'dentry')])},
            {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
            {'call': 'uselib', 'reason': set([('path', 'dentry')])},
            {'call': 'accept4',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'shmat', 'reason': set([('path', 'dentry')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'socket', 'reason': set([('path', 'dentry')])},
            {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'acct', 'reason': set([('path', 'dentry')])},
            {'call': 'open', 'reason': set([('path', 'dentry')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'dup', 'reason': set([('path', 'dentry')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'setns', 'reason': set([('path', 'dentry')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'swapon', 'reason': set([('path', 'dentry')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
            {'call': 'open_by_handle_at',
             'reason': set([('path', 'dentry')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fchmod': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'eventfd2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'swapoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'pivot_root',
```



```
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'memfd_create',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'remap_file_pages',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'dup3',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'unshare',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_create1',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl',
             'reason': set([('fd', 'flags'),
                            ('path', 'dentry'),
                            ('path', 'mnt')])},
            {'call': 'flock',
             'reason': set([('fd', 'flags'),
                            ('path', 'dentry'),
                            ('path', 'mnt')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'openat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'lookup_dcookie',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'uselib',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'accept4',
             'reason': set([('fd', 'flags'),
                            ('path', 'dentry'),
                            ('path', 'mnt')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'socketpair',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'getcwd',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'shmat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'socket',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'pipe2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```

```
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open',
             'reason': set([('fd', 'flags'),
                            ('path', 'dentry'),
                            ('path', 'mnt')])},
            {'call': 'shmdt',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'quotactl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'acct',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'open',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'dup',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'setns',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'shmctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'swapon',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'mmap_pgoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_open',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'open_by_handle_at',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fchmodat': [{'call': 'eventfd2',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'swapoff',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'pivot_root',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'memfd_create',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'remap_file_pages',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'dup3',
```



```
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'unshare',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'epoll_create1',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'epoll_ctl',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'flock',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'openat',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'lookup_dcookie',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'uselib',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'accept4',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'socketpair',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'getcwd',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'shmat',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'socket',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'pipe2',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'perf_event_open',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'shmdt',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'quotactl',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'acct',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'open',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'dup',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'setns',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'shmctl',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'swapon',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'mmap_pgoff',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'mq_open',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])},
              {'call': 'open_by_handle_at',
               'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'fchown': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
```

```
            {'call': 'remap_file_pages', 'reason': set([('path', 'dentry')])},
            {'call': 'dup3', 'reason': set([('path', 'dentry')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'unshare', 'reason': set([('path', 'dentry')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'flock',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'openat', 'reason': set([('path', 'dentry')])},
            {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
            {'call': 'uselib', 'reason': set([('path', 'dentry')])},
            {'call': 'accept4',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'shmat', 'reason': set([('path', 'dentry')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'socket', 'reason': set([('path', 'dentry')])},
            {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open',
             'reason': set([('fd', 'flags'), ('path', 'dentry')])},
            {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'acct', 'reason': set([('path', 'dentry')])},
            {'call': 'open', 'reason': set([('path', 'dentry')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'dup', 'reason': set([('path', 'dentry')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'setns', 'reason': set([('path', 'dentry')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
```



```
            {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'swapon', 'reason': set([('path', 'dentry')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
            {'call': 'open_by_handle_at',
             'reason': set([('path', 'dentry')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fchownat': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
              {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
              {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
              {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
              {'call': 'remap_file_pages',
               'reason': set([('path', 'dentry')])},
              {'call': 'dup3', 'reason': set([('path', 'dentry')])},
              {'call': 'unshare', 'reason': set([('path', 'dentry')])},
              {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
              {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
              {'call': 'flock', 'reason': set([('path', 'dentry')])},
              {'call': 'openat', 'reason': set([('path', 'dentry')])},
              {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
              {'call': 'uselib', 'reason': set([('path', 'dentry')])},
              {'call': 'accept4', 'reason': set([('path', 'dentry')])},
              {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
              {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
              {'call': 'shmat', 'reason': set([('path', 'dentry')])},
              {'call': 'socket', 'reason': set([('path', 'dentry')])},
              {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
              {'call': 'perf_event_open',
               'reason': set([('path', 'dentry')])},
              {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
              {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
              {'call': 'acct', 'reason': set([('path', 'dentry')])},
              {'call': 'open', 'reason': set([('path', 'dentry')])},
              {'call': 'dup', 'reason': set([('path', 'dentry')])},
              {'call': 'setns', 'reason': set([('path', 'dentry')])},
              {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
              {'call': 'swapon', 'reason': set([('path', 'dentry')])},
              {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
              {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
              {'call': 'open_by_handle_at',
               'reason': set([('path', 'dentry')])}],
 'fcntl': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
           {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
           {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
           {'call': 'readahead', 'reason': set([('fd', 'flags')])},
           {'call': 'getdents', 'reason': set([('fd', 'flags')])},
           {'call': 'writev', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
           {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
           {'call': 'pread64', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
            {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
            {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl',
             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
            {'call': 'flock',
             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'openat', 'reason': set([('file', 'f_mode')])},
            {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
            {'call': 'accept4',
             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'socket', 'reason': set([('file', 'f_mode')])},
            {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open',
             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
            {'call': 'open', 'reason': set([('file', 'f_mode')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
```



```
            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fcntl64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
             {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
             {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
             {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
             {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
             {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
             {'call': 'readahead', 'reason': set([('fd', 'flags')])},
             {'call': 'getdents', 'reason': set([('fd', 'flags')])},
             {'call': 'writev', 'reason': set([('fd', 'flags')])},
             {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
             {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
             {'call': 'pread64', 'reason': set([('fd', 'flags')])},
             {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
             {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
             {'call': 'remap_file_pages',
              'reason': set([('file', 'f_mode')])},
             {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
             {'call': 'read', 'reason': set([('fd', 'flags')])},
             {'call': 'fchown', 'reason': set([('fd', 'flags')])},
             {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
             {'call': 'utime', 'reason': set([('fd', 'flags')])},
             {'call': 'fsync', 'reason': set([('fd', 'flags')])},
             {'call': 'bpf', 'reason': set([('fd', 'flags')])},
             {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
             {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
             {'call': 'sendto', 'reason': set([('fd', 'flags')])},
             {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
             {'call': 'tee', 'reason': set([('fd', 'flags')])},
             {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
             {'call': 'lseek', 'reason': set([('fd', 'flags')])},
             {'call': 'connect', 'reason': set([('fd', 'flags')])},
             {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
             {'call': 'epoll_ctl',
              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
             {'call': 'flock',
              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
             {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
             {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
             {'call': 'accept4',
              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
             {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
             {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
             {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
             {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
             {'call': 'splice', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'socket', 'reason': set([('file', 'f_mode')])},
            {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
            {'call': 'fcntl',
             'reason': set([('fd', 'flags'),
                            ('flock', 'l_len'),
                            ('flock', 'l_start')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open',
             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
            {'call': 'open', 'reason': set([('file', 'f_mode')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'open_by_handle_at',
             'reason': set([('file', 'f_mode')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fgetxattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
               {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
               {'call': 'readahead', 'reason': set([('fd', 'flags')])},
               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
               {'call': 'writev', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
               {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
               {'call': 'pread64', 'reason': set([('fd', 'flags')])},
               {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
               {'call': 'read', 'reason': set([('fd', 'flags')])},
               {'call': 'fchown', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
               {'call': 'utime', 'reason': set([('fd', 'flags')])},
```



```
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
            {'call': 'flock', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch',
             'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'finit_module': [{'call': 'delete_module',
                   'reason': set([('module', 'args'),
                                  ('module', 'kp'),
                                  ('module', 'num_kp'),
                                  ('module_layout', 'base'),
                                  ('module_layout', 'size')])},
                  {'call': 'init_module',
                   'reason': set([('load_info', 'debug'),
                                  ('load_info', 'num_debug'),
                                  ('module', 'args'),
                                  ('module', 'kp'),
```

```
                                  ('module', 'num_kp'),
                                  ('module_layout', 'base'),
                                  ('module_layout', 'size')])],
 'flistxattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                {'call': 'writev', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                {'call': 'read', 'reason': set([('fd', 'flags')])},
                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
                {'call': 'utime', 'reason': set([('fd', 'flags')])},
                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                {'call': 'tee', 'reason': set([('fd', 'flags')])},
                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                {'call': 'connect', 'reason': set([('fd', 'flags')])},
                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
                {'call': 'flock', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
                {'call': 'inotify_rm_watch',
                 'reason': set([('fd', 'flags')])},
                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('fd', 'flags')])},
                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                {'call': 'splice', 'reason': set([('fd', 'flags')])},
                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                {'call': 'listen', 'reason': set([('fd', 'flags')])},
                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
```



```
                {'call': 'readv', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                {'call': 'write', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                {'call': 'bind', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'flock': [{'call': 'syncfs',
            'reason': set([('fd', 'flags'), ('super_block', 's_flags')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
           {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
           {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
           {'call': 'readahead', 'reason': set([('fd', 'flags')])},
           {'call': 'getdents', 'reason': set([('fd', 'flags')])},
           {'call': 'writev', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
           {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
           {'call': 'pread64', 'reason': set([('fd', 'flags')])},
           {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
           {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
           {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
           {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
           {'call': 'read', 'reason': set([('fd', 'flags')])},
           {'call': 'fchown', 'reason': set([('fd', 'flags')])},
           {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
           {'call': 'utime', 'reason': set([('fd', 'flags')])},
           {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
           {'call': 'fsync', 'reason': set([('fd', 'flags')])},
           {'call': 'bpf', 'reason': set([('fd', 'flags')])},
           {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
           {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
           {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
           {'call': 'sendto', 'reason': set([('fd', 'flags')])},
           {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
           {'call': 'tee', 'reason': set([('fd', 'flags')])},
           {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
           {'call': 'lseek', 'reason': set([('fd', 'flags')])},
           {'call': 'connect', 'reason': set([('fd', 'flags')])},
           {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
           {'call': 'epoll_ctl',
            'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
           {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
           {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
           {'call': 'openat', 'reason': set([('file', 'f_mode')])},
           {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
           {'call': 'accept4',
            'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
           {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
           {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
           {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
           {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
           {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
           {'call': 'splice', 'reason': set([('fd', 'flags')])},
           {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv', 'reason': set([('fd', 'flags')])},
           {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
           {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
           {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
           {'call': 'socket', 'reason': set([('file', 'f_mode')])},
           {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
           {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
           {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
```

```
2238            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
2239            {'call': 'perf_event_open',
2240             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
2241            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
2242            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
2243            {'call': 'quotactl', 'reason': set([('super_block', 's_flags')])},
2244            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
2245            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
2246            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
2247            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
2248            {'call': 'open', 'reason': set([('file', 'f_mode')])},
2249            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
2250            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
2251            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
2252            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
2253            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
2254            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
2255            {'call': 'listen', 'reason': set([('fd', 'flags')])},
2256            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
2257            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
2258            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
2259            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
2260            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
2261            {'call': 'swapon',
2262             'reason': set([('file', 'f_mode'), ('super_block', 's_flags')])},
2263            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
2264            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
2265            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
2266            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
2267            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
2268            {'call': 'readv', 'reason': set([('fd', 'flags')])},
2269            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
2270            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
2271            {'call': 'write', 'reason': set([('fd', 'flags')])},
2272            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
2273            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
2274            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
2275            {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
2276            {'call': 'bind', 'reason': set([('fd', 'flags')])},
2277            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
2278            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
2279  'fremovexattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
2280                   {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
2281                   {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
2282                   {'call': 'readahead', 'reason': set([('fd', 'flags')])},
2283                   {'call': 'getdents', 'reason': set([('fd', 'flags')])},
2284                   {'call': 'writev', 'reason': set([('fd', 'flags')])},
2285                   {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
2286                   {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
2287                   {'call': 'pread64', 'reason': set([('fd', 'flags')])},
2288                   {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
2289                   {'call': 'read', 'reason': set([('fd', 'flags')])},
2290                   {'call': 'fchown', 'reason': set([('fd', 'flags')])},
2291                   {'call': 'mq_timedreceive',
2292                    'reason': set([('fd', 'flags')])},
2293                   {'call': 'utime', 'reason': set([('fd', 'flags')])},
2294                   {'call': 'fsync', 'reason': set([('fd', 'flags')])},
2295                   {'call': 'bpf', 'reason': set([('fd', 'flags')])},
2296                   {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
2297                   {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
2298                   {'call': 'sendto', 'reason': set([('fd', 'flags')])},
2299                   {'call': 'tee', 'reason': set([('fd', 'flags')])},
2300                   {'call': 'sync_file_range',
2301                    'reason': set([('fd', 'flags')])},
2302                   {'call': 'lseek', 'reason': set([('fd', 'flags')])},
2303                   {'call': 'connect', 'reason': set([('fd', 'flags')])},
```



```
2304                   {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
2305                   {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
2306                   {'call': 'flock', 'reason': set([('fd', 'flags')])},
2307                   {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
2308                   {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
2309                   {'call': 'accept4', 'reason': set([('fd', 'flags')])},
2310                   {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
2311                   {'call': 'inotify_rm_watch',
2312                    'reason': set([('fd', 'flags')])},
2313                   {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
2314                   {'call': 'inotify_add_watch',
2315                    'reason': set([('fd', 'flags')])},
2316                   {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
2317                   {'call': 'splice', 'reason': set([('fd', 'flags')])},
2318                   {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
2319                   {'call': 'preadv', 'reason': set([('fd', 'flags')])},
2320                   {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
2321                   {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
2322                   {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
2323                   {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
2324                   {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
2325                   {'call': 'perf_event_open',
2326                    'reason': set([('fd', 'flags')])},
2327                   {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
2328                   {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
2329                   {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
2330                   {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
2331                   {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
2332                   {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
2333                   {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
2334                   {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
2335                   {'call': 'listen', 'reason': set([('fd', 'flags')])},
2336                   {'call': 'copy_file_range',
2337                    'reason': set([('fd', 'flags')])},
2338                   {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
2339                   {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
2340                   {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
2341                   {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
2342                   {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
2343                   {'call': 'llseek', 'reason': set([('fd', 'flags')])},
2344                   {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
2345                   {'call': 'readv', 'reason': set([('fd', 'flags')])},
2346                   {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
2347                   {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
2348                   {'call': 'write', 'reason': set([('fd', 'flags')])},
2349                   {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
2350                   {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
2351                   {'call': 'bind', 'reason': set([('fd', 'flags')])},
2352                   {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
2353                   {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
2354  'fsetxattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
2355                 {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
2356                 {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
2357                 {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
2358                 {'call': 'readahead', 'reason': set([('fd', 'flags')])},
2359                 {'call': 'getdents', 'reason': set([('fd', 'flags')])},
2360                 {'call': 'writev', 'reason': set([('fd', 'flags')])},
2361                 {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
2362                 {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
2363                 {'call': 'pread64', 'reason': set([('fd', 'flags')])},
2364                 {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
2365                 {'call': 'read', 'reason': set([('fd', 'flags')])},
2366                 {'call': 'fchown', 'reason': set([('fd', 'flags')])},
2367                 {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
2368                 {'call': 'utime', 'reason': set([('fd', 'flags')])},
2369                 {'call': 'fsync', 'reason': set([('fd', 'flags')])},
```

```
2370                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
2371                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
2372                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
2373                {'call': 'tee', 'reason': set([('fd', 'flags')])},
2374                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
2375                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
2376                {'call': 'connect', 'reason': set([('fd', 'flags')])},
2377                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
2378                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
2379                {'call': 'flock', 'reason': set([('fd', 'flags')])},
2380                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
2381                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
2382                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
2383                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
2384                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
2385                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
2386                {'call': 'inotify_add_watch',
2387                 'reason': set([('fd', 'flags')])},
2388                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
2389                {'call': 'splice', 'reason': set([('fd', 'flags')])},
2390                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
2391                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
2392                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
2393                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
2394                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
2395                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
2396                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
2397                {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
2398                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
2399                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
2400                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
2401                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
2402                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
2403                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
2404                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
2405                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
2406                {'call': 'listen', 'reason': set([('fd', 'flags')])},
2407                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
2408                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
2409                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
2410                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
2411                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
2412                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
2413                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
2414                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
2415                {'call': 'readv', 'reason': set([('fd', 'flags')])},
2416                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
2417                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
2418                {'call': 'write', 'reason': set([('fd', 'flags')])},
2419                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
2420                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
2421                {'call': 'bind', 'reason': set([('fd', 'flags')])},
2422                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
2423                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
2424  'fstat': [{'call': 'lstat',
2425             'reason': set([('__old_kernel_stat', 'st_ino'),
2426                            ('__old_kernel_stat', 'st_nlink')])},
2427            {'call': 'stat',
2428             'reason': set([('__old_kernel_stat', 'st_ino'),
2429                            ('__old_kernel_stat', 'st_nlink')])},
2430            {'call': 'newfstat',
2431             'reason': set([('kstat', 'dev'),
2432                            ('kstat', 'ino'),
2433                            ('kstat', 'nlink'),
2434                            ('kstat', 'rdev')])}],
2435  'fstatfs': [{'call': 'ustat',
```



```
2436             'reason': set([('kstatfs', 'f_ffree'),
2437                            ('kstatfs', 'f_files')])},
2438            {'call': 'statfs',
2439             'reason': set([('kstatfs', 'f_ffree'),
2440                            ('kstatfs', 'f_files')])},
2441            {'call': 'fstatfs64',
2442             'reason': set([('kstatfs', 'f_ffree'),
2443                            ('kstatfs', 'f_files')])},
2444            {'call': 'statfs64',
2445             'reason': set([('kstatfs', 'f_ffree'),
2446                            ('kstatfs', 'f_files')])}],
2447  'fstatfs64': [{'call': 'ustat',
2448                 'reason': set([('kstatfs', 'f_ffree'),
2449                                ('kstatfs', 'f_files')])},
2450                {'call': 'fstatfs',
2451                 'reason': set([('kstatfs', 'f_ffree'),
2452                                ('kstatfs', 'f_files')])},
2453                {'call': 'statfs',
2454                 'reason': set([('kstatfs', 'f_ffree'),
2455                                ('kstatfs', 'f_files')])},
2456                {'call': 'statfs64',
2457                 'reason': set([('kstatfs', 'f_ffree'),
2458                                ('kstatfs', 'f_files')])}],
2459  'ftruncate': [{'call': 'eventfd2',
2460                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2461                {'call': 'mq_unlink',
2462                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2463                {'call': 'swapoff',
2464                 'reason': set([('file', 'f_flags'),
2465                                ('file', 'f_mode'),
2466                                ('inode', 'i_flags'),
2467                                ('inode', 'i_sb')])},
2468                {'call': 'fchmod',
2469                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2470                {'call': 'memfd_create',
2471                 'reason': set([('file', 'f_flags'),
2472                                ('file', 'f_mode'),
2473                                ('inode', 'i_flags'),
2474                                ('inode', 'i_sb')])},
2475                {'call': 'remap_file_pages',
2476                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2477                {'call': 'dup3',
2478                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2479                {'call': 'readlinkat',
2480                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2481                {'call': 'fchown',
2482                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2483                {'call': 'mq_timedreceive',
2484                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2485                {'call': 'epoll_create1',
2486                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2487                {'call': 'epoll_ctl',
2488                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2489                {'call': 'flock',
2490                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2491                {'call': 'openat',
2492                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2493                {'call': 'uselib',
2494                 'reason': set([('file', 'f_flags'),
2495                                ('file', 'f_mode'),
2496                                ('inode', 'i_flags'),
2497                                ('inode', 'i_sb')])},
2498                {'call': 'accept4',
2499                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2500                {'call': 'socketpair',
2501                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
```

```
2502                {'call': 'fchmodat',
2503                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2504                {'call': 'inotify_add_watch',
2505                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2506                {'call': 'shmat',
2507                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2508                {'call': 'socket',
2509                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2510                {'call': 'pipe2',
2511                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2512                {'call': 'ioctl',
2513                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2514                {'call': 'perf_event_open',
2515                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2516                {'call': 'linkat',
2517                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2518                {'call': 'shmdt',
2519                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2520                {'call': 'acct',
2521                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2522                {'call': 'open',
2523                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2524                {'call': 'unlink',
2525                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2526                {'call': 'mq_getsetattr',
2527                 'reason': set([('file', 'f_flags'),
2528                                ('inode', 'i_flags'),
2529                                ('inode', 'i_sb')])},
2530                {'call': 'faccessat',
2531                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2532                {'call': 'dup',
2533                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2534                {'call': 'setns',
2535                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2536                {'call': 'mq_timedsend',
2537                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2538                {'call': 'shmctl',
2539                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2540                {'call': 'swapon',
2541                 'reason': set([('file', 'f_flags'),
2542                                ('file', 'f_mode'),
2543                                ('inode', 'i_flags'),
2544                                ('inode', 'i_sb')])},
2545                {'call': 'fchownat',
2546                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2547                {'call': 'mmap_pgoff',
2548                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2549                {'call': 'mq_notify',
2550                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2551                {'call': 'sendfile',
2552                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2553                {'call': 'mq_open',
2554                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2555                {'call': 'unlinkat',
2556                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])},
2557                {'call': 'open_by_handle_at',
2558                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
2559                {'call': 'sendfile64',
2560                 'reason': set([('inode', 'i_flags'), ('inode', 'i_sb')])}],
2561  'futex': [{'call': 'rt_sigtimedwait',
2562             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2563            {'call': 'mq_unlink',
2564             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2565            {'call': 'swapoff',
2566             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2567            {'call': 'fchmod',
```



```
2568             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2569            {'call': 'memfd_create',
2570             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2571            {'call': 'readlinkat',
2572             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2573            {'call': 'io_getevents',
2574             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2575            {'call': 'fchown',
2576             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2577            {'call': 'mq_timedreceive',
2578             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2579            {'call': 'utime',
2580             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2581            {'call': 'semtimedop',
2582             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2583            {'call': 'settimeofday',
2584             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2585            {'call': 'sched_rr_get_interval',
2586             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2587            {'call': 'timerfd_gettime',
2588             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2589            {'call': 'pselect6',
2590             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2591            {'call': 'uselib',
2592             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2593            {'call': 'fchmodat',
2594             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2595            {'call': 'inotify_add_watch',
2596             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2597            {'call': 'timer_settime',
2598             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2599            {'call': 'ftruncate',
2600             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2601            {'call': 'timer_gettime',
2602             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2603            {'call': 'ioctl',
2604             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2605            {'call': 'linkat',
2606             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2607            {'call': 'stime',
2608             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2609            {'call': 'futimesat',
2610             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2611            {'call': 'poll',
2612             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2613            {'call': 'select',
2614             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2615            {'call': 'unlink',
2616             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2617            {'call': 'nanosleep',
2618             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2619            {'call': 'mq_getsetattr',
2620             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2621            {'call': 'faccessat',
2622             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2623            {'call': 'mq_timedsend',
2624             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2625            {'call': 'swapon',
2626             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2627            {'call': 'epoll_wait',
2628             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2629            {'call': 'fchownat',
2630             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2631            {'call': 'fstat',
2632             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2633            {'call': 'timerfd_settime',
```

```
2634              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2635             {'call': 'mq_notify',
2636              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2637             {'call': 'sendfile',
2638              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2639             {'call': 'newfstat',
2640              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2641             {'call': 'clock_nanosleep',
2642              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2643             {'call': 'unlinkat',
2644              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2645             {'call': 'recvmmsg',
2646              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2647             {'call': 'sendfile64',
2648              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
2649             {'call': 'ppoll',
2650              'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])}],
2651  'futimesat': [{'call': 'rt_sigtimedwait',
2652                 'reason': set([('timespec', 'tv_nsec')])},
2653                {'call': 'mq_unlink',
2654                 'reason': set([('timespec', 'tv_nsec')])},
2655                {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
2656                {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
2657                {'call': 'memfd_create',
2658                 'reason': set([('timespec', 'tv_nsec')])},
2659                {'call': 'readlinkat',
2660                 'reason': set([('timespec', 'tv_nsec')])},
2661                {'call': 'io_getevents',
2662                 'reason': set([('timespec', 'tv_nsec')])},
2663                {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
2664                {'call': 'waitid', 'reason': set([('timeval', 'tv_usec')])},
2665                {'call': 'mq_timedreceive',
2666                 'reason': set([('timespec', 'tv_nsec')])},
2667                {'call': 'utime', 'reason': set([('timespec', 'tv_nsec')])},
2668                {'call': 'semtimedop',
2669                 'reason': set([('timespec', 'tv_nsec')])},
2670                {'call': 'settimeofday',
2671                 'reason': set([('timespec', 'tv_nsec'),
2672                                ('timeval', 'tv_usec')])},
2673                {'call': 'sched_rr_get_interval',
2674                 'reason': set([('timespec', 'tv_nsec')])},
2675                {'call': 'timerfd_gettime',
2676                 'reason': set([('timespec', 'tv_nsec')])},
2677                {'call': 'adjtimex', 'reason': set([('timeval', 'tv_usec')])},
2678                {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
2679                {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
2680                {'call': 'getitimer', 'reason': set([('timeval', 'tv_usec')])},
2681                {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
2682                {'call': 'inotify_add_watch',
2683                 'reason': set([('timespec', 'tv_nsec')])},
2684                {'call': 'timer_settime',
2685                 'reason': set([('timespec', 'tv_nsec')])},
2686                {'call': 'ftruncate',
2687                 'reason': set([('timespec', 'tv_nsec')])},
2688                {'call': 'timer_gettime',
2689                 'reason': set([('timespec', 'tv_nsec')])},
2690                {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
2691                {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
2692                {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
2693                {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
2694                {'call': 'select',
2695                 'reason': set([('timespec', 'tv_nsec'),
2696                                ('timeval', 'tv_usec')])},
2697                {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
2698                {'call': 'nanosleep',
2699                 'reason': set([('timespec', 'tv_nsec')])},
```



```
2700                {'call': 'mq_getsetattr',
2701                 'reason': set([('timespec', 'tv_nsec')])},
2702                {'call': 'faccessat',
2703                 'reason': set([('timespec', 'tv_nsec')])},
2704                {'call': 'mq_timedsend',
2705                 'reason': set([('timespec', 'tv_nsec')])},
2706                {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
2707                {'call': 'wait4', 'reason': set([('timeval', 'tv_usec')])},
2708                {'call': 'epoll_wait',
2709                 'reason': set([('timespec', 'tv_nsec')])},
2710                {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
2711                {'call': 'getrusage', 'reason': set([('timeval', 'tv_usec')])},
2712                {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
2713                {'call': 'timerfd_settime',
2714                 'reason': set([('timespec', 'tv_nsec')])},
2715                {'call': 'setitimer', 'reason': set([('timeval', 'tv_usec')])},
2716                {'call': 'mq_notify',
2717                 'reason': set([('timespec', 'tv_nsec')])},
2718                {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
2719                {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
2720                {'call': 'clock_nanosleep',
2721                 'reason': set([('timespec', 'tv_nsec')])},
2722                {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
2723                {'call': 'clock_adjtime',
2724                 'reason': set([('timeval', 'tv_usec')])},
2725                {'call': 'alarm', 'reason': set([('timeval', 'tv_usec')])},
2726                {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
2727                {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
2728                {'call': 'sendfile64',
2729                 'reason': set([('timespec', 'tv_nsec')])},
2730                {'call': 'ppoll',
2731                 'reason': set([('timespec', 'tv_nsec'),
2732                                ('timeval', 'tv_usec')])}],
2733  'get_mempolicy': [{'call': 'keyctl',
2734                     'reason': set([('task_struct', 'il_prev'),
2735                                    ('task_struct', 'mempolicy')])},
2736                    {'call': 'rt_sigtimedwait',
2737                     'reason': set([('task_struct', 'il_prev'),
2738                                    ('task_struct', 'mempolicy')])},
2739                    {'call': 'msgrcv',
2740                     'reason': set([('task_struct', 'il_prev'),
2741                                    ('task_struct', 'mempolicy')])},
2742                    {'call': 'kill',
2743                     'reason': set([('task_struct', 'il_prev'),
2744                                    ('task_struct', 'mempolicy')])},
2745                    {'call': 'sched_getaffinity',
2746                     'reason': set([('task_struct', 'il_prev'),
2747                                    ('task_struct', 'mempolicy')])},
2748                    {'call': 'sched_setparam',
2749                     'reason': set([('task_struct', 'il_prev'),
2750                                    ('task_struct', 'mempolicy')])},
2751                    {'call': 'ioprio_set',
2752                     'reason': set([('task_struct', 'il_prev'),
2753                                    ('task_struct', 'mempolicy')])},
2754                    {'call': 'remap_file_pages',
2755                     'reason': set([('vm_area_struct', 'vm_ops')])},
2756                    {'call': 'getppid',
2757                     'reason': set([('task_struct', 'il_prev'),
2758                                    ('task_struct', 'mempolicy')])},
2759                    {'call': 'mq_timedreceive',
2760                     'reason': set([('task_struct', 'il_prev'),
2761                                    ('task_struct', 'mempolicy')])},
2762                    {'call': 'capget',
2763                     'reason': set([('task_struct', 'il_prev'),
2764                                    ('task_struct', 'mempolicy')])},
2765                    {'call': 'sched_setaffinity',
```

```
2766                     'reason': set([('task_struct', 'il_prev'),
2767                                    ('task_struct', 'mempolicy')])},
2768                 {'call': 'signal',
2769                  'reason': set([('task_struct', 'il_prev'),
2770                                 ('task_struct', 'mempolicy')])},
2771                 {'call': 'semtimedop',
2772                  'reason': set([('task_struct', 'il_prev'),
2773                                 ('task_struct', 'mempolicy')])},
2774                 {'call': 'umount',
2775                  'reason': set([('task_struct', 'il_prev'),
2776                                 ('task_struct', 'mempolicy')])},
2777                 {'call': 'sched_rr_get_interval',
2778                  'reason': set([('task_struct', 'il_prev'),
2779                                 ('task_struct', 'mempolicy')])},
2780                 {'call': 'rt_sigprocmask',
2781                  'reason': set([('task_struct', 'il_prev'),
2782                                 ('task_struct', 'mempolicy')])},
2783                 {'call': 'setsid',
2784                  'reason': set([('task_struct', 'il_prev'),
2785                                 ('task_struct', 'mempolicy')])},
2786                 {'call': 'sigaltstack',
2787                  'reason': set([('task_struct', 'il_prev'),
2788                                 ('task_struct', 'mempolicy')])},
2789                 {'call': 'sched_setattr',
2790                  'reason': set([('task_struct', 'il_prev'),
2791                                 ('task_struct', 'mempolicy')])},
2792                 {'call': 'migrate_pages',
2793                  'reason': set([('task_struct', 'il_prev'),
2794                                 ('task_struct', 'mempolicy')])},
2795                 {'call': 'getitimer',
2796                  'reason': set([('task_struct', 'il_prev'),
2797                                 ('task_struct', 'mempolicy')])},
2798                 {'call': 'setpgid',
2799                  'reason': set([('task_struct', 'il_prev'),
2800                                 ('task_struct', 'mempolicy')])},
2801                 {'call': 'getsid',
2802                  'reason': set([('task_struct', 'il_prev'),
2803                                 ('task_struct', 'mempolicy')])},
2804                 {'call': 'prlimit64',
2805                  'reason': set([('task_struct', 'il_prev'),
2806                                 ('task_struct', 'mempolicy')])},
2807                 {'call': 'set_mempolicy',
2808                  'reason': set([('mempolicy', 'mode'),
2809                                 ('task_struct', 'il_prev'),
2810                                 ('task_struct', 'mempolicy')])},
2811                 {'call': 'perf_event_open',
2812                  'reason': set([('task_struct', 'il_prev'),
2813                                 ('task_struct', 'mempolicy')])},
2814                 {'call': 'shmdt',
2815                  'reason': set([('vm_area_struct', 'vm_ops')])},
2816                 {'call': 'rt_sigaction',
2817                  'reason': set([('task_struct', 'il_prev'),
2818                                 ('task_struct', 'mempolicy')])},
2819                 {'call': 'getpgid',
2820                  'reason': set([('task_struct', 'il_prev'),
2821                                 ('task_struct', 'mempolicy')])},
2822                 {'call': 'brk',
2823                  'reason': set([('vm_area_struct', 'vm_ops')])},
2824                 {'call': 'getpriority',
2825                  'reason': set([('task_struct', 'il_prev'),
2826                                 ('task_struct', 'mempolicy')])},
2827                 {'call': 'sigaction',
2828                  'reason': set([('task_struct', 'il_prev'),
2829                                 ('task_struct', 'mempolicy')])},
2830                 {'call': 'setns',
2831                  'reason': set([('task_struct', 'il_prev'),
```



```
2832                                 ('task_struct', 'mempolicy')])},
2833                 {'call': 'fork',
2834                  'reason': set([('task_struct', 'il_prev'),
2835                                 ('task_struct', 'mempolicy')])},
2836                 {'call': 'get_robust_list',
2837                  'reason': set([('task_struct', 'il_prev'),
2838                                 ('task_struct', 'mempolicy')])},
2839                 {'call': 'mq_timedsend',
2840                  'reason': set([('task_struct', 'il_prev'),
2841                                 ('task_struct', 'mempolicy')])},
2842                 {'call': 'sched_getscheduler',
2843                  'reason': set([('task_struct', 'il_prev'),
2844                                 ('task_struct', 'mempolicy')])},
2845                 {'call': 'ptrace',
2846                  'reason': set([('task_struct', 'il_prev'),
2847                                 ('task_struct', 'mempolicy')])},
2848                 {'call': 'munlockall',
2849                  'reason': set([('vm_area_struct', 'vm_ops')])},
2850                 {'call': 'pkey_mprotect',
2851                  'reason': set([('vm_area_struct', 'vm_ops')])},
2852                 {'call': 'madvise',
2853                  'reason': set([('vm_area_struct', 'vm_ops')])},
2854                 {'call': 'sched_getattr',
2855                  'reason': set([('task_struct', 'il_prev'),
2856                                 ('task_struct', 'mempolicy')])},
2857                 {'call': 'getrusage',
2858                  'reason': set([('task_struct', 'il_prev'),
2859                                 ('task_struct', 'mempolicy')])},
2860                 {'call': 'sched_setscheduler',
2861                  'reason': set([('task_struct', 'il_prev'),
2862                                 ('task_struct', 'mempolicy')])},
2863                 {'call': 'setitimer',
2864                  'reason': set([('task_struct', 'il_prev'),
2865                                 ('task_struct', 'mempolicy')])},
2866                 {'call': 'ioprio_get',
2867                  'reason': set([('task_struct', 'il_prev'),
2868                                 ('task_struct', 'mempolicy')])},
2869                 {'call': 'vfork',
2870                  'reason': set([('task_struct', 'il_prev'),
2871                                 ('task_struct', 'mempolicy')])},
2872                 {'call': 'mprotect',
2873                  'reason': set([('vm_area_struct', 'vm_ops')])},
2874                 {'call': 'mremap',
2875                  'reason': set([('vm_area_struct', 'vm_ops')])},
2876                 {'call': 'mbind', 'reason': set([('mempolicy', 'mode')])},
2877                 {'call': 'prctl',
2878                  'reason': set([('task_struct', 'il_prev'),
2879                                 ('task_struct', 'mempolicy'),
2880                                 ('vm_area_struct', 'vm_ops')])},
2881                 {'call': 'move_pages',
2882                  'reason': set([('task_struct', 'il_prev'),
2883                                 ('task_struct', 'mempolicy')])},
2884                 {'call': 'munlock',
2885                  'reason': set([('vm_area_struct', 'vm_ops')])},
2886                 {'call': 'setpriority',
2887                  'reason': set([('task_struct', 'il_prev'),
2888                                 ('task_struct', 'mempolicy')])},
2889                 {'call': 'mincore',
2890                  'reason': set([('vm_area_struct', 'vm_ops')])},
2891                 {'call': 'clone',
2892                  'reason': set([('task_struct', 'il_prev'),
2893                                 ('task_struct', 'mempolicy')])},
2894                 {'call': 'sched_getparam',
2895                  'reason': set([('task_struct', 'il_prev'),
2896                                 ('task_struct', 'mempolicy')])},
2897                 {'call': 'mlockall',
```

```
                    'reason': set([('vm_area_struct', 'vm_ops')])}],
 'getcwd': [{'call': 'eventfd2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mq_unlink',
             'reason': set([('dentry', 'd_parent'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'swapoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'pivot_root',
             'reason': set([('dentry', 'd_parent'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'memfd_create',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'remap_file_pages',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'dup3',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'unshare',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'umount', 'reason': set([('vfsmount', 'mnt_root')])},
            {'call': 'mkdirat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'epoll_create1',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'epoll_ctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'flock',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'openat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'lookup_dcookie',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'uselib',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'renameat2', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'accept4',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'socketpair',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'ftruncate', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'shmat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mknodat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'socket',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'symlinkat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'pipe2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'perf_event_open',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'linkat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'shmdt',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'quotactl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'acct',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'open',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'unlink', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'rmdir', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'dup',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```



```
            {'call': 'setns',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'shmctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'swapon',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mmap_pgoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mq_open',
             'reason': set([('dentry', 'd_parent'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'unlinkat', 'reason': set([('dentry', 'd_parent')])},
            {'call': 'open_by_handle_at',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'getdents': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
              {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
              {'call': 'readahead', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_getaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'writev', 'reason': set([('fd', 'flags')])},
              {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_setparam',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
              {'call': 'pread64', 'reason': set([('fd', 'flags')])},
              {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
              {'call': 'ioprio_set',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'read', 'reason': set([('fd', 'flags')])},
              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchown', 'reason': set([('fd', 'flags')])},
              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_timedreceive',
               'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'utime', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_setaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fsync', 'reason': set([('fd', 'flags')])},
              {'call': 'bpf', 'reason': set([('fd', 'flags')])},
              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'semtimedop',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
              {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
              {'call': 'sendto', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'tee', 'reason': set([('fd', 'flags')])},
              {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
              {'call': 'lseek', 'reason': set([('fd', 'flags')])},
              {'call': 'connect', 'reason': set([('fd', 'flags')])},
              {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
              {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
              {'call': 'flock', 'reason': set([('fd', 'flags')])},
              {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
```

```
3030                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
3031                {'call': 'rt_sigprocmask',
3032                 'reason': set([('mm_segment_t', 'seg')])},
3033                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
3034                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
3035                {'call': 'sigaltstack',
3036                 'reason': set([('mm_segment_t', 'seg')])},
3037                {'call': 'sched_setattr',
3038                 'reason': set([('mm_segment_t', 'seg')])},
3039                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
3040                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
3041                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
3042                {'call': 'migrate_pages',
3043                 'reason': set([('mm_segment_t', 'seg')])},
3044                {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
3045                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
3046                {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
3047                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
3048                {'call': 'splice', 'reason': set([('fd', 'flags')])},
3049                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
3050                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
3051                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
3052                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
3053                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
3054                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
3055                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
3056                {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
3057                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
3058                {'call': 'perf_event_open',
3059                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
3060                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
3061                {'call': 'rt_sigaction',
3062                 'reason': set([('mm_segment_t', 'seg')])},
3063                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
3064                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
3065                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
3066                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
3067                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
3068                {'call': 'getpriority',
3069                 'reason': set([('mm_segment_t', 'seg')])},
3070                {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
3071                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
3072                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
3073                {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
3074                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
3075                {'call': 'listen', 'reason': set([('fd', 'flags')])},
3076                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
3077                {'call': 'get_robust_list',
3078                 'reason': set([('mm_segment_t', 'seg')])},
3079                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
3080                {'call': 'mq_timedsend',
3081                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
3082                {'call': 'sched_getscheduler',
3083                 'reason': set([('mm_segment_t', 'seg')])},
3084                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
3085                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
3086                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
3087                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
3088                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
3089                {'call': 'sched_getattr',
3090                 'reason': set([('mm_segment_t', 'seg')])},
3091                {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
3092                {'call': 'sched_setscheduler',
3093                 'reason': set([('mm_segment_t', 'seg')])},
3094                {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
3095                {'call': 'ioprio_get',
```



```
3096                 'reason': set([('mm_segment_t', 'seg')])},
3097                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
3098                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
3099                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
3100                {'call': 'readv', 'reason': set([('fd', 'flags')])},
3101                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
3102                {'call': 'move_pages',
3103                 'reason': set([('mm_segment_t', 'seg')])},
3104                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
3105                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
3106                {'call': 'write', 'reason': set([('fd', 'flags')])},
3107                {'call': 'setpriority',
3108                 'reason': set([('mm_segment_t', 'seg')])},
3109                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
3110                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
3111                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
3112                {'call': 'sched_getparam',
3113                 'reason': set([('mm_segment_t', 'seg')])},
3114                {'call': 'bind', 'reason': set([('fd', 'flags')])},
3115                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
3116                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
3117  'getdents64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
3118                 {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
3119                 {'call': 'rt_sigtimedwait',
3120                  'reason': set([('mm_segment_t', 'seg')])},
3121                 {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
3122                 {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
3123                 {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
3124                 {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
3125                 {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
3126                 {'call': 'readahead', 'reason': set([('fd', 'flags')])},
3127                 {'call': 'getdents', 'reason': set([('fd', 'flags')])},
3128                 {'call': 'sched_getaffinity',
3129                  'reason': set([('mm_segment_t', 'seg')])},
3130                 {'call': 'writev', 'reason': set([('fd', 'flags')])},
3131                 {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
3132                 {'call': 'sched_setparam',
3133                  'reason': set([('mm_segment_t', 'seg')])},
3134                 {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
3135                 {'call': 'pread64', 'reason': set([('fd', 'flags')])},
3136                 {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
3137                 {'call': 'ioprio_set',
3138                  'reason': set([('mm_segment_t', 'seg')])},
3139                 {'call': 'read', 'reason': set([('fd', 'flags')])},
3140                 {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
3141                 {'call': 'fchown', 'reason': set([('fd', 'flags')])},
3142                 {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
3143                 {'call': 'mq_timedreceive',
3144                  'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
3145                 {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
3146                 {'call': 'utime', 'reason': set([('fd', 'flags')])},
3147                 {'call': 'sched_setaffinity',
3148                  'reason': set([('mm_segment_t', 'seg')])},
3149                 {'call': 'fsync', 'reason': set([('fd', 'flags')])},
3150                 {'call': 'bpf', 'reason': set([('fd', 'flags')])},
3151                 {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
3152                 {'call': 'semtimedop',
3153                  'reason': set([('mm_segment_t', 'seg')])},
3154                 {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
3155                 {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
3156                 {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
3157                 {'call': 'sendto', 'reason': set([('fd', 'flags')])},
3158                 {'call': 'sched_rr_get_interval',
3159                  'reason': set([('mm_segment_t', 'seg')])},
3160                 {'call': 'tee', 'reason': set([('fd', 'flags')])},
3161                 {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
```

```
3162                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
3163                {'call': 'connect', 'reason': set([('fd', 'flags')])},
3164                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
3165                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
3166                {'call': 'flock', 'reason': set([('fd', 'flags')])},
3167                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
3168                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
3169                {'call': 'rt_sigprocmask',
3170                 'reason': set([('mm_segment_t', 'seg')])},
3171                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
3172                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
3173                {'call': 'sigaltstack',
3174                 'reason': set([('mm_segment_t', 'seg')])},
3175                {'call': 'sched_setattr',
3176                 'reason': set([('mm_segment_t', 'seg')])},
3177                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
3178                {'call': 'inotify_rm_watch',
3179                 'reason': set([('fd', 'flags')])},
3180                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
3181                {'call': 'migrate_pages',
3182                 'reason': set([('mm_segment_t', 'seg')])},
3183                {'call': 'getitimer',
3184                 'reason': set([('mm_segment_t', 'seg')])},
3185                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
3186                {'call': 'inotify_add_watch',
3187                 'reason': set([('fd', 'flags')])},
3188                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
3189                {'call': 'splice', 'reason': set([('fd', 'flags')])},
3190                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
3191                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
3192                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
3193                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
3194                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
3195                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
3196                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
3197                {'call': 'prlimit64',
3198                 'reason': set([('mm_segment_t', 'seg')])},
3199                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
3200                {'call': 'perf_event_open',
3201                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
3202                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
3203                {'call': 'rt_sigaction',
3204                 'reason': set([('mm_segment_t', 'seg')])},
3205                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
3206                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
3207                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
3208                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
3209                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
3210                {'call': 'getpriority',
3211                 'reason': set([('mm_segment_t', 'seg')])},
3212                {'call': 'sigaction',
3213                 'reason': set([('mm_segment_t', 'seg')])},
3214                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
3215                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
3216                {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
3217                {'call': 'listen', 'reason': set([('fd', 'flags')])},
3218                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
3219                {'call': 'get_robust_list',
3220                 'reason': set([('mm_segment_t', 'seg')])},
3221                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
3222                {'call': 'mq_timedsend',
3223                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
3224                {'call': 'sched_getscheduler',
3225                 'reason': set([('mm_segment_t', 'seg')])},
3226                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
3227                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
```



```
3228                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
3229                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
3230                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
3231                {'call': 'sched_getattr',
3232                 'reason': set([('mm_segment_t', 'seg')])},
3233                {'call': 'getrusage',
3234                 'reason': set([('mm_segment_t', 'seg')])},
3235                {'call': 'sched_setscheduler',
3236                 'reason': set([('mm_segment_t', 'seg')])},
3237                {'call': 'setitimer',
3238                 'reason': set([('mm_segment_t', 'seg')])},
3239                {'call': 'ioprio_get',
3240                 'reason': set([('mm_segment_t', 'seg')])},
3241                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
3242                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
3243                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
3244                {'call': 'readv', 'reason': set([('fd', 'flags')])},
3245                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
3246                {'call': 'move_pages',
3247                 'reason': set([('mm_segment_t', 'seg')])},
3248                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
3249                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
3250                {'call': 'write', 'reason': set([('fd', 'flags')])},
3251                {'call': 'setpriority',
3252                 'reason': set([('mm_segment_t', 'seg')])},
3253                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
3254                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
3255                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
3256                {'call': 'sched_getparam',
3257                 'reason': set([('mm_segment_t', 'seg')])},
3258                {'call': 'bind', 'reason': set([('fd', 'flags')])},
3259                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
3260                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
3261  'getegid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
3262              {'call': 'rt_sigtimedwait',
3263               'reason': set([('task_struct', 'cred')])},
3264              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3265              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3266              {'call': 'sched_getaffinity',
3267               'reason': set([('task_struct', 'cred')])},
3268              {'call': 'sched_setparam',
3269               'reason': set([('task_struct', 'cred')])},
3270              {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
3271              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
3272              {'call': 'mq_timedreceive',
3273               'reason': set([('task_struct', 'cred')])},
3274              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3275              {'call': 'sched_setaffinity',
3276               'reason': set([('task_struct', 'cred')])},
3277              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
3278              {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
3279              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
3280              {'call': 'sched_rr_get_interval',
3281               'reason': set([('task_struct', 'cred')])},
3282              {'call': 'rt_sigprocmask',
3283               'reason': set([('task_struct', 'cred')])},
3284              {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
3285              {'call': 'sigaltstack',
3286               'reason': set([('task_struct', 'cred')])},
3287              {'call': 'sched_setattr',
3288               'reason': set([('task_struct', 'cred')])},
3289              {'call': 'migrate_pages',
3290               'reason': set([('task_struct', 'cred')])},
3291              {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
3292              {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
3293              {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
```

```
3294              {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
3295              {'call': 'perf_event_open',
3296               'reason': set([('task_struct', 'cred')])},
3297              {'call': 'rt_sigaction',
3298               'reason': set([('task_struct', 'cred')])},
3299              {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
3300              {'call': 'getpriority',
3301               'reason': set([('task_struct', 'cred')])},
3302              {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
3303              {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3304              {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3305              {'call': 'get_robust_list',
3306               'reason': set([('task_struct', 'cred')])},
3307              {'call': 'mq_timedsend',
3308               'reason': set([('task_struct', 'cred')])},
3309              {'call': 'sched_getscheduler',
3310               'reason': set([('task_struct', 'cred')])},
3311              {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3312              {'call': 'sched_getattr',
3313               'reason': set([('task_struct', 'cred')])},
3314              {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
3315              {'call': 'sched_setscheduler',
3316               'reason': set([('task_struct', 'cred')])},
3317              {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
3318              {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
3319              {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
3320              {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3321              {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
3322              {'call': 'setpriority',
3323               'reason': set([('task_struct', 'cred')])},
3324              {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3325              {'call': 'sched_getparam',
3326               'reason': set([('task_struct', 'cred')])}],
3327  'getegid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
3328                {'call': 'rt_sigtimedwait',
3329                 'reason': set([('task_struct', 'cred')])},
3330                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3331                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3332                {'call': 'sched_getaffinity',
3333                 'reason': set([('task_struct', 'cred')])},
3334                {'call': 'sched_setparam',
3335                 'reason': set([('task_struct', 'cred')])},
3336                {'call': 'ioprio_set',
3337                 'reason': set([('task_struct', 'cred')])},
3338                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
3339                {'call': 'mq_timedreceive',
3340                 'reason': set([('task_struct', 'cred')])},
3341                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3342                {'call': 'sched_setaffinity',
3343                 'reason': set([('task_struct', 'cred')])},
3344                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
3345                {'call': 'semtimedop',
3346                 'reason': set([('task_struct', 'cred')])},
3347                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
3348                {'call': 'sched_rr_get_interval',
3349                 'reason': set([('task_struct', 'cred')])},
3350                {'call': 'rt_sigprocmask',
3351                 'reason': set([('task_struct', 'cred')])},
3352                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
3353                {'call': 'sigaltstack',
3354                 'reason': set([('task_struct', 'cred')])},
3355                {'call': 'sched_setattr',
3356                 'reason': set([('task_struct', 'cred')])},
3357                {'call': 'migrate_pages',
3358                 'reason': set([('task_struct', 'cred')])},
3359                {'call': 'getitimer',
```



```
3360                 'reason': set([('task_struct', 'cred')])},
3361                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
3362                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
3363                {'call': 'prlimit64',
3364                 'reason': set([('task_struct', 'cred')])},
3365                {'call': 'perf_event_open',
3366                 'reason': set([('task_struct', 'cred')])},
3367                {'call': 'rt_sigaction',
3368                 'reason': set([('task_struct', 'cred')])},
3369                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
3370                {'call': 'getpriority',
3371                 'reason': set([('task_struct', 'cred')])},
3372                {'call': 'sigaction',
3373                 'reason': set([('task_struct', 'cred')])},
3374                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3375                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3376                {'call': 'get_robust_list',
3377                 'reason': set([('task_struct', 'cred')])},
3378                {'call': 'mq_timedsend',
3379                 'reason': set([('task_struct', 'cred')])},
3380                {'call': 'sched_getscheduler',
3381                 'reason': set([('task_struct', 'cred')])},
3382                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3383                {'call': 'sched_getattr',
3384                 'reason': set([('task_struct', 'cred')])},
3385                {'call': 'getrusage',
3386                 'reason': set([('task_struct', 'cred')])},
3387                {'call': 'sched_setscheduler',
3388                 'reason': set([('task_struct', 'cred')])},
3389                {'call': 'setitimer',
3390                 'reason': set([('task_struct', 'cred')])},
3391                {'call': 'ioprio_get',
3392                 'reason': set([('task_struct', 'cred')])},
3393                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
3394                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3395                {'call': 'move_pages',
3396                 'reason': set([('task_struct', 'cred')])},
3397                {'call': 'setpriority',
3398                 'reason': set([('task_struct', 'cred')])},
3399                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3400                {'call': 'sched_getparam',
3401                 'reason': set([('task_struct', 'cred')])}],
3402  'geteuid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
3403              {'call': 'rt_sigtimedwait',
3404               'reason': set([('task_struct', 'cred')])},
3405              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3406              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3407              {'call': 'sched_getaffinity',
3408               'reason': set([('task_struct', 'cred')])},
3409              {'call': 'sched_setparam',
3410               'reason': set([('task_struct', 'cred')])},
3411              {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
3412              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
3413              {'call': 'mq_timedreceive',
3414               'reason': set([('task_struct', 'cred')])},
3415              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3416              {'call': 'sched_setaffinity',
3417               'reason': set([('task_struct', 'cred')])},
3418              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
3419              {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
3420              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
3421              {'call': 'sched_rr_get_interval',
3422               'reason': set([('task_struct', 'cred')])},
3423              {'call': 'rt_sigprocmask',
3424               'reason': set([('task_struct', 'cred')])},
3425              {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
```

```
3426             {'call': 'sigaltstack',
3427              'reason': set([('task_struct', 'cred')])},
3428             {'call': 'sched_setattr',
3429              'reason': set([('task_struct', 'cred')])},
3430             {'call': 'migrate_pages',
3431              'reason': set([('task_struct', 'cred')])},
3432             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
3433             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
3434             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
3435             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
3436             {'call': 'perf_event_open',
3437              'reason': set([('task_struct', 'cred')])},
3438             {'call': 'rt_sigaction',
3439              'reason': set([('task_struct', 'cred')])},
3440             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
3441             {'call': 'getpriority',
3442              'reason': set([('task_struct', 'cred')])},
3443             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
3444             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3445             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3446             {'call': 'get_robust_list',
3447              'reason': set([('task_struct', 'cred')])},
3448             {'call': 'mq_timedsend',
3449              'reason': set([('task_struct', 'cred')])},
3450             {'call': 'sched_getscheduler',
3451              'reason': set([('task_struct', 'cred')])},
3452             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3453             {'call': 'sched_getattr',
3454              'reason': set([('task_struct', 'cred')])},
3455             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
3456             {'call': 'sched_setscheduler',
3457              'reason': set([('task_struct', 'cred')])},
3458             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
3459             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
3460             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
3461             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3462             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
3463             {'call': 'setpriority',
3464              'reason': set([('task_struct', 'cred')])},
3465             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3466             {'call': 'sched_getparam',
3467              'reason': set([('task_struct', 'cred')])}],
3468  'geteuid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
3469                {'call': 'rt_sigtimedwait',
3470                 'reason': set([('task_struct', 'cred')])},
3471                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3472                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3473                {'call': 'sched_getaffinity',
3474                 'reason': set([('task_struct', 'cred')])},
3475                {'call': 'sched_setparam',
3476                 'reason': set([('task_struct', 'cred')])},
3477                {'call': 'ioprio_set',
3478                 'reason': set([('task_struct', 'cred')])},
3479                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
3480                {'call': 'mq_timedreceive',
3481                 'reason': set([('task_struct', 'cred')])},
3482                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3483                {'call': 'sched_setaffinity',
3484                 'reason': set([('task_struct', 'cred')])},
3485                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
3486                {'call': 'semtimedop',
3487                 'reason': set([('task_struct', 'cred')])},
3488                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
3489                {'call': 'sched_rr_get_interval',
3490                 'reason': set([('task_struct', 'cred')])},
3491                {'call': 'rt_sigprocmask',
```



```
3492                 'reason': set([('task_struct', 'cred')])},
3493                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
3494                {'call': 'sigaltstack',
3495                 'reason': set([('task_struct', 'cred')])},
3496                {'call': 'sched_setattr',
3497                 'reason': set([('task_struct', 'cred')])},
3498                {'call': 'migrate_pages',
3499                 'reason': set([('task_struct', 'cred')])},
3500                {'call': 'getitimer',
3501                 'reason': set([('task_struct', 'cred')])},
3502                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
3503                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
3504                {'call': 'prlimit64',
3505                 'reason': set([('task_struct', 'cred')])},
3506                {'call': 'perf_event_open',
3507                 'reason': set([('task_struct', 'cred')])},
3508                {'call': 'rt_sigaction',
3509                 'reason': set([('task_struct', 'cred')])},
3510                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
3511                {'call': 'getpriority',
3512                 'reason': set([('task_struct', 'cred')])},
3513                {'call': 'sigaction',
3514                 'reason': set([('task_struct', 'cred')])},
3515                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3516                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3517                {'call': 'get_robust_list',
3518                 'reason': set([('task_struct', 'cred')])},
3519                {'call': 'mq_timedsend',
3520                 'reason': set([('task_struct', 'cred')])},
3521                {'call': 'sched_getscheduler',
3522                 'reason': set([('task_struct', 'cred')])},
3523                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3524                {'call': 'sched_getattr',
3525                 'reason': set([('task_struct', 'cred')])},
3526                {'call': 'getrusage',
3527                 'reason': set([('task_struct', 'cred')])},
3528                {'call': 'sched_setscheduler',
3529                 'reason': set([('task_struct', 'cred')])},
3530                {'call': 'setitimer',
3531                 'reason': set([('task_struct', 'cred')])},
3532                {'call': 'ioprio_get',
3533                 'reason': set([('task_struct', 'cred')])},
3534                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
3535                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3536                {'call': 'move_pages',
3537                 'reason': set([('task_struct', 'cred')])},
3538                {'call': 'setpriority',
3539                 'reason': set([('task_struct', 'cred')])},
3540                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3541                {'call': 'sched_getparam',
3542                 'reason': set([('task_struct', 'cred')])}],
3543  'getgid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
3544             {'call': 'rt_sigtimedwait',
3545              'reason': set([('task_struct', 'cred')])},
3546             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3547             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3548             {'call': 'sched_getaffinity',
3549              'reason': set([('task_struct', 'cred')])},
3550             {'call': 'sched_setparam',
3551              'reason': set([('task_struct', 'cred')])},
3552             {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
3553             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
3554             {'call': 'mq_timedreceive',
3555              'reason': set([('task_struct', 'cred')])},
3556             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3557             {'call': 'sched_setaffinity',
```

```
3558              'reason': set([('task_struct', 'cred')])},
3559             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
3560             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
3561             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
3562             {'call': 'sched_rr_get_interval',
3563              'reason': set([('task_struct', 'cred')])},
3564             {'call': 'rt_sigprocmask',
3565              'reason': set([('task_struct', 'cred')])},
3566             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
3567             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
3568             {'call': 'sched_setattr',
3569              'reason': set([('task_struct', 'cred')])},
3570             {'call': 'migrate_pages',
3571              'reason': set([('task_struct', 'cred')])},
3572             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
3573             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
3574             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
3575             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
3576             {'call': 'perf_event_open',
3577              'reason': set([('task_struct', 'cred')])},
3578             {'call': 'rt_sigaction',
3579              'reason': set([('task_struct', 'cred')])},
3580             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
3581             {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
3582             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
3583             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3584             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3585             {'call': 'get_robust_list',
3586              'reason': set([('task_struct', 'cred')])},
3587             {'call': 'mq_timedsend',
3588              'reason': set([('task_struct', 'cred')])},
3589             {'call': 'sched_getscheduler',
3590              'reason': set([('task_struct', 'cred')])},
3591             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3592             {'call': 'sched_getattr',
3593              'reason': set([('task_struct', 'cred')])},
3594             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
3595             {'call': 'sched_setscheduler',
3596              'reason': set([('task_struct', 'cred')])},
3597             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
3598             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
3599             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
3600             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3601             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
3602             {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
3603             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3604             {'call': 'sched_getparam',
3605              'reason': set([('task_struct', 'cred')])}],
3606  'getgid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
3607               {'call': 'rt_sigtimedwait',
3608                'reason': set([('task_struct', 'cred')])},
3609               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3610               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3611               {'call': 'sched_getaffinity',
3612                'reason': set([('task_struct', 'cred')])},
3613               {'call': 'sched_setparam',
3614                'reason': set([('task_struct', 'cred')])},
3615               {'call': 'ioprio_set',
3616                'reason': set([('task_struct', 'cred')])},
3617               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
3618               {'call': 'mq_timedreceive',
3619                'reason': set([('task_struct', 'cred')])},
3620               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3621               {'call': 'sched_setaffinity',
3622                'reason': set([('task_struct', 'cred')])},
3623               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
```



```
3624             {'call': 'semtimedop',
3625              'reason': set([('task_struct', 'cred')])},
3626             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
3627             {'call': 'sched_rr_get_interval',
3628              'reason': set([('task_struct', 'cred')])},
3629             {'call': 'rt_sigprocmask',
3630              'reason': set([('task_struct', 'cred')])},
3631             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
3632             {'call': 'sigaltstack',
3633              'reason': set([('task_struct', 'cred')])},
3634             {'call': 'sched_setattr',
3635              'reason': set([('task_struct', 'cred')])},
3636             {'call': 'migrate_pages',
3637              'reason': set([('task_struct', 'cred')])},
3638             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
3639             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
3640             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
3641             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
3642             {'call': 'perf_event_open',
3643              'reason': set([('task_struct', 'cred')])},
3644             {'call': 'rt_sigaction',
3645              'reason': set([('task_struct', 'cred')])},
3646             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
3647             {'call': 'getpriority',
3648              'reason': set([('task_struct', 'cred')])},
3649             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
3650             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3651             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3652             {'call': 'get_robust_list',
3653              'reason': set([('task_struct', 'cred')])},
3654             {'call': 'mq_timedsend',
3655              'reason': set([('task_struct', 'cred')])},
3656             {'call': 'sched_getscheduler',
3657              'reason': set([('task_struct', 'cred')])},
3658             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3659             {'call': 'sched_getattr',
3660              'reason': set([('task_struct', 'cred')])},
3661             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
3662             {'call': 'sched_setscheduler',
3663              'reason': set([('task_struct', 'cred')])},
3664             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
3665             {'call': 'ioprio_get',
3666              'reason': set([('task_struct', 'cred')])},
3667             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
3668             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3669             {'call': 'move_pages',
3670              'reason': set([('task_struct', 'cred')])},
3671             {'call': 'setpriority',
3672              'reason': set([('task_struct', 'cred')])},
3673             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3674             {'call': 'sched_getparam',
3675              'reason': set([('task_struct', 'cred')])}],
3676  'getgroups': [{'call': 'keyctl',
3677                 'reason': set([('cred', 'group_info'),
3678                                ('task_struct', 'cred')])},
3679                {'call': 'rt_sigtimedwait',
3680                 'reason': set([('task_struct', 'cred')])},
3681                {'call': 'setfsuid', 'reason': set([('cred', 'group_info')])},
3682                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3683                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3684                {'call': 'getresuid16',
3685                 'reason': set([('cred', 'group_info')])},
3686                {'call': 'getresgid', 'reason': set([('cred', 'group_info')])},
3687                {'call': 'sched_getaffinity',
3688                 'reason': set([('task_struct', 'cred')])},
3689                {'call': 'sched_setparam',
```

```
3690                 'reason': set([('task_struct', 'cred')])},
3691                {'call': 'setgid', 'reason': set([('cred', 'group_info')])},
3692                {'call': 'ioprio_set',
3693                 'reason': set([('cred', 'group_info'),
3694                                ('task_struct', 'cred')])},
3695                {'call': 'capset', 'reason': set([('cred', 'group_info')])},
3696                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
3697                {'call': 'mq_timedreceive',
3698                 'reason': set([('task_struct', 'cred')])},
3699                {'call': 'getresgid16',
3700                 'reason': set([('cred', 'group_info')])},
3701                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3702                {'call': 'sched_setaffinity',
3703                 'reason': set([('cred', 'group_info'),
3704                                ('task_struct', 'cred')])},
3705                {'call': 'setfsgid', 'reason': set([('cred', 'group_info')])},
3706                {'call': 'unshare', 'reason': set([('cred', 'group_info')])},
3707                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
3708                {'call': 'setreuid', 'reason': set([('cred', 'group_info')])},
3709                {'call': 'semtimedop',
3710                 'reason': set([('task_struct', 'cred')])},
3711                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
3712                {'call': 'sched_rr_get_interval',
3713                 'reason': set([('task_struct', 'cred')])},
3714                {'call': 'epoll_create1',
3715                 'reason': set([('cred', 'group_info')])},
3716                {'call': 'getresuid', 'reason': set([('cred', 'group_info')])},
3717                {'call': 'rt_sigprocmask',
3718                 'reason': set([('task_struct', 'cred')])},
3719                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
3720                {'call': 'sigaltstack',
3721                 'reason': set([('task_struct', 'cred')])},
3722                {'call': 'sched_setattr',
3723                 'reason': set([('task_struct', 'cred')])},
3724                {'call': 'migrate_pages',
3725                 'reason': set([('cred', 'group_info'),
3726                                ('task_struct', 'cred')])},
3727                {'call': 'getitimer',
3728                 'reason': set([('task_struct', 'cred')])},
3729                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
3730                {'call': 'setresgid', 'reason': set([('cred', 'group_info')])},
3731                {'call': 'setregid', 'reason': set([('cred', 'group_info')])},
3732                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
3733                {'call': 'prlimit64',
3734                 'reason': set([('cred', 'group_info'),
3735                                ('task_struct', 'cred')])},
3736                {'call': 'perf_event_open',
3737                 'reason': set([('task_struct', 'cred')])},
3738                {'call': 'getgroups16',
3739                 'reason': set([('cred', 'group_info')])},
3740                {'call': 'rt_sigaction',
3741                 'reason': set([('task_struct', 'cred')])},
3742                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
3743                {'call': 'getpriority',
3744                 'reason': set([('cred', 'group_info'),
3745                                ('task_struct', 'cred')])},
3746                {'call': 'sigaction',
3747                 'reason': set([('task_struct', 'cred')])},
3748                {'call': 'faccessat', 'reason': set([('cred', 'group_info')])},
3749                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3750                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3751                {'call': 'get_robust_list',
3752                 'reason': set([('task_struct', 'cred')])},
3753                {'call': 'mq_timedsend',
3754                 'reason': set([('task_struct', 'cred')])},
3755                {'call': 'sched_getscheduler',
```



```
3756                 'reason': set([('task_struct', 'cred')])},
3757                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3758                {'call': 'sched_getattr',
3759                 'reason': set([('task_struct', 'cred')])},
3760                {'call': 'getrusage',
3761                 'reason': set([('task_struct', 'cred')])},
3762                {'call': 'sched_setscheduler',
3763                 'reason': set([('task_struct', 'cred')])},
3764                {'call': 'setresuid', 'reason': set([('cred', 'group_info')])},
3765                {'call': 'setitimer',
3766                 'reason': set([('task_struct', 'cred')])},
3767                {'call': 'ioprio_get',
3768                 'reason': set([('cred', 'group_info'),
3769                                ('task_struct', 'cred')])},
3770                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
3771                {'call': 'setuid', 'reason': set([('cred', 'group_info')])},
3772                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3773                {'call': 'move_pages',
3774                 'reason': set([('task_struct', 'cred')])},
3775                {'call': 'setpriority',
3776                 'reason': set([('cred', 'group_info'),
3777                                ('task_struct', 'cred')])},
3778                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3779                {'call': 'sched_getparam',
3780                 'reason': set([('task_struct', 'cred')])}],
3781  'getgroups16': [{'call': 'keyctl',
3782                   'reason': set([('cred', 'group_info'),
3783                                  ('task_struct', 'cred')])},
3784                  {'call': 'rt_sigtimedwait',
3785                   'reason': set([('task_struct', 'cred')])},
3786                  {'call': 'setfsuid',
3787                   'reason': set([('cred', 'group_info')])},
3788                  {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
3789                  {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
3790                  {'call': 'getresuid16',
3791                   'reason': set([('cred', 'group_info')])},
3792                  {'call': 'getresgid',
3793                   'reason': set([('cred', 'group_info')])},
3794                  {'call': 'sched_getaffinity',
3795                   'reason': set([('task_struct', 'cred')])},
3796                  {'call': 'sched_setparam',
3797                   'reason': set([('task_struct', 'cred')])},
3798                  {'call': 'setgid', 'reason': set([('cred', 'group_info')])},
3799                  {'call': 'ioprio_set',
3800                   'reason': set([('cred', 'group_info'),
3801                                  ('task_struct', 'cred')])},
3802                  {'call': 'capset', 'reason': set([('cred', 'group_info')])},
3803                  {'call': 'getppid',
3804                   'reason': set([('task_struct', 'cred')])},
3805                  {'call': 'mq_timedreceive',
3806                   'reason': set([('task_struct', 'cred')])},
3807                  {'call': 'getresgid16',
3808                   'reason': set([('cred', 'group_info')])},
3809                  {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
3810                  {'call': 'sched_setaffinity',
3811                   'reason': set([('cred', 'group_info'),
3812                                  ('task_struct', 'cred')])},
3813                  {'call': 'setfsgid',
3814                   'reason': set([('cred', 'group_info')])},
3815                  {'call': 'unshare', 'reason': set([('cred', 'group_info')])},
3816                  {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
3817                  {'call': 'setreuid',
3818                   'reason': set([('cred', 'group_info')])},
3819                  {'call': 'semtimedop',
3820                   'reason': set([('task_struct', 'cred')])},
3821                  {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
```

```
3822                 {'call': 'sched_rr_get_interval',
3823                  'reason': set([('task_struct', 'cred')])},
3824                 {'call': 'epoll_create1',
3825                  'reason': set([('cred', 'group_info')])},
3826                 {'call': 'getresuid',
3827                  'reason': set([('cred', 'group_info')])},
3828                 {'call': 'rt_sigprocmask',
3829                  'reason': set([('task_struct', 'cred')])},
3830                 {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
3831                 {'call': 'sigaltstack',
3832                  'reason': set([('task_struct', 'cred')])},
3833                 {'call': 'sched_setattr',
3834                  'reason': set([('task_struct', 'cred')])},
3835                 {'call': 'migrate_pages',
3836                  'reason': set([('cred', 'group_info'),
3837                                 ('task_struct', 'cred')])},
3838                 {'call': 'getitimer',
3839                  'reason': set([('task_struct', 'cred')])},
3840                 {'call': 'setpgid',
3841                  'reason': set([('task_struct', 'cred')])},
3842                 {'call': 'setresgid',
3843                  'reason': set([('cred', 'group_info')])},
3844                 {'call': 'setregid',
3845                  'reason': set([('cred', 'group_info')])},
3846                 {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
3847                 {'call': 'prlimit64',
3848                  'reason': set([('cred', 'group_info'),
3849                                 ('task_struct', 'cred')])},
3850                 {'call': 'perf_event_open',
3851                  'reason': set([('task_struct', 'cred')])},
3852                 {'call': 'rt_sigaction',
3853                  'reason': set([('task_struct', 'cred')])},
3854                 {'call': 'getpgid',
3855                  'reason': set([('task_struct', 'cred')])},
3856                 {'call': 'getpriority',
3857                  'reason': set([('cred', 'group_info'),
3858                                 ('task_struct', 'cred')])},
3859                 {'call': 'sigaction',
3860                  'reason': set([('task_struct', 'cred')])},
3861                 {'call': 'faccessat',
3862                  'reason': set([('cred', 'group_info')])},
3863                 {'call': 'setgroups16',
3864                  'reason': set([('group_info', 'ngroups')])},
3865                 {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
3866                 {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
3867                 {'call': 'get_robust_list',
3868                  'reason': set([('task_struct', 'cred')])},
3869                 {'call': 'mq_timedsend',
3870                  'reason': set([('task_struct', 'cred')])},
3871                 {'call': 'sched_getscheduler',
3872                  'reason': set([('task_struct', 'cred')])},
3873                 {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
3874                 {'call': 'sched_getattr',
3875                  'reason': set([('task_struct', 'cred')])},
3876                 {'call': 'getrusage',
3877                  'reason': set([('task_struct', 'cred')])},
3878                 {'call': 'sched_setscheduler',
3879                  'reason': set([('task_struct', 'cred')])},
3880                 {'call': 'setresuid',
3881                  'reason': set([('cred', 'group_info')])},
3882                 {'call': 'setitimer',
3883                  'reason': set([('task_struct', 'cred')])},
3884                 {'call': 'ioprio_get',
3885                  'reason': set([('cred', 'group_info'),
3886                                 ('task_struct', 'cred')])},
3887                 {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
```



```
3888                 {'call': 'setuid', 'reason': set([('cred', 'group_info')])},
3889                 {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
3890                 {'call': 'move_pages',
3891                  'reason': set([('task_struct', 'cred')])},
3892                 {'call': 'getgroups',
3893                  'reason': set([('cred', 'group_info')])},
3894                 {'call': 'setpriority',
3895                  'reason': set([('cred', 'group_info'),
3896                                 ('task_struct', 'cred')])},
3897                 {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
3898                 {'call': 'setgroups',
3899                  'reason': set([('group_info', 'ngroups')])},
3900                 {'call': 'sched_getparam',
3901                  'reason': set([('task_struct', 'cred')])}],
3902  'getitimer': [{'call': 'timer_create',
3903                 'reason': set([('signal_struct', 'it_real_incr')])},
3904                {'call': 'exit_group',
3905                 'reason': set([('signal_struct', 'it_real_incr')])},
3906                {'call': 'setitimer',
3907                 'reason': set([('signal_struct', 'it_real_incr')])}],
3908  'getppid': [{'call': 'keyctl',
3909               'reason': set([('task_struct', 'real_parent')])},
3910              {'call': 'rt_sigtimedwait',
3911               'reason': set([('task_struct', 'real_parent')])},
3912              {'call': 'msgrcv',
3913               'reason': set([('task_struct', 'real_parent')])},
3914              {'call': 'kill',
3915               'reason': set([('task_struct', 'real_parent')])},
3916              {'call': 'sched_getaffinity',
3917               'reason': set([('task_struct', 'real_parent')])},
3918              {'call': 'sched_setparam',
3919               'reason': set([('task_struct', 'real_parent')])},
3920              {'call': 'ioprio_set',
3921               'reason': set([('task_struct', 'real_parent')])},
3922              {'call': 'mq_timedreceive',
3923               'reason': set([('task_struct', 'real_parent')])},
3924              {'call': 'capget',
3925               'reason': set([('task_struct', 'real_parent')])},
3926              {'call': 'sched_setaffinity',
3927               'reason': set([('task_struct', 'real_parent')])},
3928              {'call': 'signal',
3929               'reason': set([('task_struct', 'real_parent')])},
3930              {'call': 'semtimedop',
3931               'reason': set([('task_struct', 'real_parent')])},
3932              {'call': 'umount',
3933               'reason': set([('task_struct', 'real_parent')])},
3934              {'call': 'sched_rr_get_interval',
3935               'reason': set([('task_struct', 'real_parent')])},
3936              {'call': 'rt_sigprocmask',
3937               'reason': set([('task_struct', 'real_parent')])},
3938              {'call': 'setsid',
3939               'reason': set([('task_struct', 'real_parent')])},
3940              {'call': 'sigaltstack',
3941               'reason': set([('task_struct', 'real_parent')])},
3942              {'call': 'sched_setattr',
3943               'reason': set([('task_struct', 'real_parent')])},
3944              {'call': 'migrate_pages',
3945               'reason': set([('task_struct', 'real_parent')])},
3946              {'call': 'getitimer',
3947               'reason': set([('task_struct', 'real_parent')])},
3948              {'call': 'setpgid',
3949               'reason': set([('task_struct', 'real_parent')])},
3950              {'call': 'getsid',
3951               'reason': set([('task_struct', 'real_parent')])},
3952              {'call': 'prlimit64',
3953               'reason': set([('task_struct', 'real_parent')])},
```

```
3954             {'call': 'perf_event_open',
3955              'reason': set([('task_struct', 'real_parent')])},
3956             {'call': 'rt_sigaction',
3957              'reason': set([('task_struct', 'real_parent')])},
3958             {'call': 'getpgid',
3959              'reason': set([('task_struct', 'real_parent')])},
3960             {'call': 'getpriority',
3961              'reason': set([('task_struct', 'real_parent')])},
3962             {'call': 'sigaction',
3963              'reason': set([('task_struct', 'real_parent')])},
3964             {'call': 'setns',
3965              'reason': set([('task_struct', 'real_parent')])},
3966             {'call': 'fork',
3967              'reason': set([('task_struct', 'real_parent')])},
3968             {'call': 'get_robust_list',
3969              'reason': set([('task_struct', 'real_parent')])},
3970             {'call': 'mq_timedsend',
3971              'reason': set([('task_struct', 'real_parent')])},
3972             {'call': 'sched_getscheduler',
3973              'reason': set([('task_struct', 'real_parent')])},
3974             {'call': 'ptrace',
3975              'reason': set([('task_struct', 'real_parent')])},
3976             {'call': 'sched_getattr',
3977              'reason': set([('task_struct', 'real_parent')])},
3978             {'call': 'getrusage',
3979              'reason': set([('task_struct', 'real_parent')])},
3980             {'call': 'sched_setscheduler',
3981              'reason': set([('task_struct', 'real_parent')])},
3982             {'call': 'setitimer',
3983              'reason': set([('task_struct', 'real_parent')])},
3984             {'call': 'ioprio_get',
3985              'reason': set([('task_struct', 'real_parent')])},
3986             {'call': 'vfork',
3987              'reason': set([('task_struct', 'real_parent')])},
3988             {'call': 'prctl',
3989              'reason': set([('task_struct', 'real_parent')])},
3990             {'call': 'move_pages',
3991              'reason': set([('task_struct', 'real_parent')])},
3992             {'call': 'setpriority',
3993              'reason': set([('task_struct', 'real_parent')])},
3994             {'call': 'clone',
3995              'reason': set([('task_struct', 'real_parent')])},
3996             {'call': 'sched_getparam',
3997              'reason': set([('task_struct', 'real_parent')])}],
3998  'getpriority': [{'call': 'keyctl',
3999                   'reason': set([('task_struct', 'cred'),
4000                                  ('task_struct', 'real_cred')])},
4001                  {'call': 'rt_sigtimedwait',
4002                   'reason': set([('task_struct', 'cred'),
4003                                  ('task_struct', 'real_cred')])},
4004                  {'call': 'msgrcv',
4005                   'reason': set([('task_struct', 'cred'),
4006                                  ('task_struct', 'real_cred')])},
4007                  {'call': 'kill',
4008                   'reason': set([('task_struct', 'cred'),
4009                                  ('task_struct', 'real_cred')])},
4010                  {'call': 'sched_getaffinity',
4011                   'reason': set([('task_struct', 'cred'),
4012                                  ('task_struct', 'real_cred')])},
4013                  {'call': 'sched_setparam',
4014                   'reason': set([('task_struct', 'cred'),
4015                                  ('task_struct', 'real_cred')])},
4016                  {'call': 'ioprio_set',
4017                   'reason': set([('task_struct', 'cred'),
4018                                  ('task_struct', 'real_cred')])},
4019                  {'call': 'getppid',
```



```
4020                   'reason': set([('task_struct', 'cred'),
4021                                  ('task_struct', 'real_cred')])},
4022                  {'call': 'mq_timedreceive',
4023                   'reason': set([('task_struct', 'cred'),
4024                                  ('task_struct', 'real_cred')])},
4025                  {'call': 'capget',
4026                   'reason': set([('task_struct', 'cred'),
4027                                  ('task_struct', 'real_cred')])},
4028                  {'call': 'sched_setaffinity',
4029                   'reason': set([('task_struct', 'cred'),
4030                                  ('task_struct', 'real_cred')])},
4031                  {'call': 'signal',
4032                   'reason': set([('task_struct', 'cred'),
4033                                  ('task_struct', 'real_cred')])},
4034                  {'call': 'setreuid', 'reason': set([('cred', 'uid')])},
4035                  {'call': 'semtimedop',
4036                   'reason': set([('task_struct', 'cred'),
4037                                  ('task_struct', 'real_cred')])},
4038                  {'call': 'umount',
4039                   'reason': set([('task_struct', 'cred'),
4040                                  ('task_struct', 'real_cred')])},
4041                  {'call': 'sched_rr_get_interval',
4042                   'reason': set([('task_struct', 'cred'),
4043                                  ('task_struct', 'real_cred')])},
4044                  {'call': 'rt_sigprocmask',
4045                   'reason': set([('task_struct', 'cred'),
4046                                  ('task_struct', 'real_cred')])},
4047                  {'call': 'setsid',
4048                   'reason': set([('task_struct', 'cred'),
4049                                  ('task_struct', 'real_cred')])},
4050                  {'call': 'sigaltstack',
4051                   'reason': set([('task_struct', 'cred'),
4052                                  ('task_struct', 'real_cred')])},
4053                  {'call': 'sched_setattr',
4054                   'reason': set([('task_struct', 'cred'),
4055                                  ('task_struct', 'real_cred')])},
4056                  {'call': 'migrate_pages',
4057                   'reason': set([('task_struct', 'cred'),
4058                                  ('task_struct', 'real_cred')])},
4059                  {'call': 'getitimer',
4060                   'reason': set([('task_struct', 'cred'),
4061                                  ('task_struct', 'real_cred')])},
4062                  {'call': 'setpgid',
4063                   'reason': set([('task_struct', 'cred'),
4064                                  ('task_struct', 'real_cred')])},
4065                  {'call': 'getsid',
4066                   'reason': set([('task_struct', 'cred'),
4067                                  ('task_struct', 'real_cred')])},
4068                  {'call': 'prlimit64',
4069                   'reason': set([('task_struct', 'cred'),
4070                                  ('task_struct', 'real_cred')])},
4071                  {'call': 'perf_event_open',
4072                   'reason': set([('task_struct', 'cred'),
4073                                  ('task_struct', 'real_cred')])},
4074                  {'call': 'rt_sigaction',
4075                   'reason': set([('task_struct', 'cred'),
4076                                  ('task_struct', 'real_cred')])},
4077                  {'call': 'getpgid',
4078                   'reason': set([('task_struct', 'cred'),
4079                                  ('task_struct', 'real_cred')])},
4080                  {'call': 'sigaction',
4081                   'reason': set([('task_struct', 'cred'),
4082                                  ('task_struct', 'real_cred')])},
4083                  {'call': 'setns',
4084                   'reason': set([('task_struct', 'cred'),
4085                                  ('task_struct', 'real_cred')])},
```

```
                {'call': 'fork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setresuid', 'reason': set([('cred', 'uid')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setuid', 'reason': set([('cred', 'uid')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])}],
 'getresgid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getaffinity',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setparam',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ioprio_set',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedreceive',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setaffinity',
```



```
                'reason': set([('task_struct', 'cred')])},
               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
               {'call': 'semtimedop',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_rr_get_interval',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigprocmask',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaltstack',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'migrate_pages',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'prlimit64',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'perf_event_open',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'getpriority',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
               {'call': 'get_robust_list',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedsend',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getscheduler',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getrusage',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setscheduler',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ioprio_get',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
               {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
               {'call': 'move_pages',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setpriority',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getparam',
                'reason': set([('task_struct', 'cred')])}],
 'getresgid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
                 {'call': 'rt_sigtimedwait',
                  'reason': set([('task_struct', 'cred')])},
                 {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
                 {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
                 {'call': 'sched_getaffinity',
                  'reason': set([('task_struct', 'cred')])},
```

```
                {'call': 'sched_setparam',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ioprio_set',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'cred')])},
                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred')])},
```



```
                 {'call': 'setpriority',
                  'reason': set([('task_struct', 'cred')])},
                 {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
                 {'call': 'sched_getparam',
                  'reason': set([('task_struct', 'cred')])}],
 'getresuid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getaffinity',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setparam',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ioprio_set',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedreceive',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setaffinity',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
               {'call': 'semtimedop',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_rr_get_interval',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigprocmask',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaltstack',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'migrate_pages',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getitimer',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'prlimit64',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'perf_event_open',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'rt_sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
               {'call': 'getpriority',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sigaction',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
               {'call': 'get_robust_list',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'mq_timedsend',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getscheduler',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_getattr',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'getrusage',
                'reason': set([('task_struct', 'cred')])},
               {'call': 'sched_setscheduler',
```

```
4350                 'reason': set([('task_struct', 'cred')])},
4351                {'call': 'setitimer',
4352                 'reason': set([('task_struct', 'cred')])},
4353                {'call': 'ioprio_get',
4354                 'reason': set([('task_struct', 'cred')])},
4355                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
4356                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
4357                {'call': 'move_pages',
4358                 'reason': set([('task_struct', 'cred')])},
4359                {'call': 'setpriority',
4360                 'reason': set([('task_struct', 'cred')])},
4361                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
4362                {'call': 'sched_getparam',
4363                 'reason': set([('task_struct', 'cred')])}],
4364  'getresuid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
4365                  {'call': 'rt_sigtimedwait',
4366                   'reason': set([('task_struct', 'cred')])},
4367                  {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
4368                  {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
4369                  {'call': 'sched_getaffinity',
4370                   'reason': set([('task_struct', 'cred')])},
4371                  {'call': 'sched_setparam',
4372                   'reason': set([('task_struct', 'cred')])},
4373                  {'call': 'ioprio_set',
4374                   'reason': set([('task_struct', 'cred')])},
4375                  {'call': 'getppid',
4376                   'reason': set([('task_struct', 'cred')])},
4377                  {'call': 'mq_timedreceive',
4378                   'reason': set([('task_struct', 'cred')])},
4379                  {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
4380                  {'call': 'sched_setaffinity',
4381                   'reason': set([('task_struct', 'cred')])},
4382                  {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
4383                  {'call': 'semtimedop',
4384                   'reason': set([('task_struct', 'cred')])},
4385                  {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
4386                  {'call': 'sched_rr_get_interval',
4387                   'reason': set([('task_struct', 'cred')])},
4388                  {'call': 'rt_sigprocmask',
4389                   'reason': set([('task_struct', 'cred')])},
4390                  {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
4391                  {'call': 'sigaltstack',
4392                   'reason': set([('task_struct', 'cred')])},
4393                  {'call': 'sched_setattr',
4394                   'reason': set([('task_struct', 'cred')])},
4395                  {'call': 'migrate_pages',
4396                   'reason': set([('task_struct', 'cred')])},
4397                  {'call': 'getitimer',
4398                   'reason': set([('task_struct', 'cred')])},
4399                  {'call': 'setpgid',
4400                   'reason': set([('task_struct', 'cred')])},
4401                  {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
4402                  {'call': 'prlimit64',
4403                   'reason': set([('task_struct', 'cred')])},
4404                  {'call': 'perf_event_open',
4405                   'reason': set([('task_struct', 'cred')])},
4406                  {'call': 'rt_sigaction',
4407                   'reason': set([('task_struct', 'cred')])},
4408                  {'call': 'getpgid',
4409                   'reason': set([('task_struct', 'cred')])},
4410                  {'call': 'getpriority',
4411                   'reason': set([('task_struct', 'cred')])},
4412                  {'call': 'sigaction',
4413                   'reason': set([('task_struct', 'cred')])},
4414                  {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
4415                  {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
```



```
4416                 {'call': 'get_robust_list',
4417                  'reason': set([('task_struct', 'cred')])},
4418                 {'call': 'mq_timedsend',
4419                  'reason': set([('task_struct', 'cred')])},
4420                 {'call': 'sched_getscheduler',
4421                  'reason': set([('task_struct', 'cred')])},
4422                 {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
4423                 {'call': 'sched_getattr',
4424                  'reason': set([('task_struct', 'cred')])},
4425                 {'call': 'getrusage',
4426                  'reason': set([('task_struct', 'cred')])},
4427                 {'call': 'sched_setscheduler',
4428                  'reason': set([('task_struct', 'cred')])},
4429                 {'call': 'setitimer',
4430                  'reason': set([('task_struct', 'cred')])},
4431                 {'call': 'ioprio_get',
4432                  'reason': set([('task_struct', 'cred')])},
4433                 {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
4434                 {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
4435                 {'call': 'move_pages',
4436                  'reason': set([('task_struct', 'cred')])},
4437                 {'call': 'setpriority',
4438                  'reason': set([('task_struct', 'cred')])},
4439                 {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
4440                 {'call': 'sched_getparam',
4441                  'reason': set([('task_struct', 'cred')])}],
4442  'getrlimit': [{'call': 'setrlimit',
4443                 'reason': set([('rlimit', 'rlim_cur'),
4444                                ('rlimit', 'rlim_max')])},
4445                {'call': 'old_getrlimit',
4446                 'reason': set([('rlimit', 'rlim_cur'),
4447                                ('rlimit', 'rlim_max')])},
4448                {'call': 'prlimit64',
4449                 'reason': set([('rlimit', 'rlim_cur'),
4450                                ('rlimit', 'rlim_max')])}],
4451  'getrusage': [{'call': 'timer_create',
4452                 'reason': set([('signal_struct', 'maxrss')])},
4453                {'call': 'exit_group',
4454                 'reason': set([('signal_struct', 'maxrss')])}],
4455  'getsockopt': [{'call': 'accept4',
4456                  'reason': set([('proto_ops', 'compat_getsockopt')])}],
4457  'getuid': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
4458             {'call': 'rt_sigtimedwait',
4459              'reason': set([('task_struct', 'cred')])},
4460             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
4461             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
4462             {'call': 'sched_getaffinity',
4463              'reason': set([('task_struct', 'cred')])},
4464             {'call': 'sched_setparam',
4465              'reason': set([('task_struct', 'cred')])},
4466             {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
4467             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
4468             {'call': 'mq_timedreceive',
4469              'reason': set([('task_struct', 'cred')])},
4470             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
4471             {'call': 'sched_setaffinity',
4472              'reason': set([('task_struct', 'cred')])},
4473             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
4474             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
4475             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
4476             {'call': 'sched_rr_get_interval',
4477              'reason': set([('task_struct', 'cred')])},
4478             {'call': 'rt_sigprocmask',
4479              'reason': set([('task_struct', 'cred')])},
4480             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
4481             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
```

```
4482             {'call': 'sched_setattr',
4483              'reason': set([('task_struct', 'cred')])},
4484             {'call': 'migrate_pages',
4485              'reason': set([('task_struct', 'cred')])},
4486             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
4487             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
4488             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
4489             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
4490             {'call': 'perf_event_open',
4491              'reason': set([('task_struct', 'cred')])},
4492             {'call': 'rt_sigaction',
4493              'reason': set([('task_struct', 'cred')])},
4494             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
4495             {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
4496             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
4497             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
4498             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
4499             {'call': 'get_robust_list',
4500              'reason': set([('task_struct', 'cred')])},
4501             {'call': 'mq_timedsend',
4502              'reason': set([('task_struct', 'cred')])},
4503             {'call': 'sched_getscheduler',
4504              'reason': set([('task_struct', 'cred')])},
4505             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
4506             {'call': 'sched_getattr',
4507              'reason': set([('task_struct', 'cred')])},
4508             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
4509             {'call': 'sched_setscheduler',
4510              'reason': set([('task_struct', 'cred')])},
4511             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
4512             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
4513             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
4514             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
4515             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
4516             {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
4517             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
4518             {'call': 'sched_getparam',
4519              'reason': set([('task_struct', 'cred')])}],
4520  'getuid16': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
4521               {'call': 'rt_sigtimedwait',
4522                'reason': set([('task_struct', 'cred')])},
4523               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
4524               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
4525               {'call': 'sched_getaffinity',
4526                'reason': set([('task_struct', 'cred')])},
4527               {'call': 'sched_setparam',
4528                'reason': set([('task_struct', 'cred')])},
4529               {'call': 'ioprio_set',
4530                'reason': set([('task_struct', 'cred')])},
4531               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
4532               {'call': 'mq_timedreceive',
4533                'reason': set([('task_struct', 'cred')])},
4534               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
4535               {'call': 'sched_setaffinity',
4536                'reason': set([('task_struct', 'cred')])},
4537               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
4538               {'call': 'semtimedop',
4539                'reason': set([('task_struct', 'cred')])},
4540               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
4541               {'call': 'sched_rr_get_interval',
4542                'reason': set([('task_struct', 'cred')])},
4543               {'call': 'rt_sigprocmask',
4544                'reason': set([('task_struct', 'cred')])},
4545               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
4546               {'call': 'sigaltstack',
4547                'reason': set([('task_struct', 'cred')])},
```



```
4548             {'call': 'sched_setattr',
4549              'reason': set([('task_struct', 'cred')])},
4550             {'call': 'migrate_pages',
4551              'reason': set([('task_struct', 'cred')])},
4552             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
4553             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
4554             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
4555             {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
4556             {'call': 'perf_event_open',
4557              'reason': set([('task_struct', 'cred')])},
4558             {'call': 'rt_sigaction',
4559              'reason': set([('task_struct', 'cred')])},
4560             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
4561             {'call': 'getpriority',
4562              'reason': set([('task_struct', 'cred')])},
4563             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
4564             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
4565             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
4566             {'call': 'get_robust_list',
4567              'reason': set([('task_struct', 'cred')])},
4568             {'call': 'mq_timedsend',
4569              'reason': set([('task_struct', 'cred')])},
4570             {'call': 'sched_getscheduler',
4571              'reason': set([('task_struct', 'cred')])},
4572             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
4573             {'call': 'sched_getattr',
4574              'reason': set([('task_struct', 'cred')])},
4575             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
4576             {'call': 'sched_setscheduler',
4577              'reason': set([('task_struct', 'cred')])},
4578             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
4579             {'call': 'ioprio_get',
4580              'reason': set([('task_struct', 'cred')])},
4581             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
4582             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
4583             {'call': 'move_pages',
4584              'reason': set([('task_struct', 'cred')])},
4585             {'call': 'setpriority',
4586              'reason': set([('task_struct', 'cred')])},
4587             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
4588             {'call': 'sched_getparam',
4589              'reason': set([('task_struct', 'cred')])}],
4590  'getxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
4591               {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
4592               {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
4593               {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
4594               {'call': 'remap_file_pages',
4595                'reason': set([('path', 'dentry')])},
4596               {'call': 'dup3', 'reason': set([('path', 'dentry')])},
4597               {'call': 'unshare', 'reason': set([('path', 'dentry')])},
4598               {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
4599               {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
4600               {'call': 'flock', 'reason': set([('path', 'dentry')])},
4601               {'call': 'openat', 'reason': set([('path', 'dentry')])},
4602               {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
4603               {'call': 'uselib', 'reason': set([('path', 'dentry')])},
4604               {'call': 'accept4', 'reason': set([('path', 'dentry')])},
4605               {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
4606               {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
4607               {'call': 'shmat', 'reason': set([('path', 'dentry')])},
4608               {'call': 'socket', 'reason': set([('path', 'dentry')])},
4609               {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
4610               {'call': 'perf_event_open',
4611                'reason': set([('path', 'dentry')])},
4612               {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
4613               {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
```

```
4614              {'call': 'acct', 'reason': set([('path', 'dentry')])},
4615              {'call': 'open', 'reason': set([('path', 'dentry')])},
4616              {'call': 'dup', 'reason': set([('path', 'dentry')])},
4617              {'call': 'setns', 'reason': set([('path', 'dentry')])},
4618              {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
4619              {'call': 'swapon', 'reason': set([('path', 'dentry')])},
4620              {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
4621              {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
4622              {'call': 'open_by_handle_at',
4623               'reason': set([('path', 'dentry')])}],
4624  'init_module': [{'call': 'delete_module',
4625                   'reason': set([('module', 'args'),
4626                                  ('module', 'kp'),
4627                                  ('module', 'num_kp'),
4628                                  ('module_layout', 'base'),
4629                                  ('module_layout', 'size')])},
4630                  {'call': 'finit_module',
4631                   'reason': set([('load_info', 'debug'),
4632                                  ('load_info', 'hdr'),
4633                                  ('load_info', 'len'),
4634                                  ('load_info', 'num_debug'),
4635                                  ('module', 'args'),
4636                                  ('module', 'kp'),
4637                                  ('module', 'num_kp'),
4638                                  ('module_layout', 'base'),
4639                                  ('module_layout', 'size')])}],
4640  'inotify_add_watch': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
4641                        {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
4642                        {'call': 'eventfd2',
4643                         'reason': set([('file', 'f_op')])},
4644                        {'call': 'mq_unlink',
4645                         'reason': set([('dentry', 'd_inode')])},
4646                        {'call': 'pwritev64',
4647                         'reason': set([('fd', 'flags')])},
4648                        {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
4649                        {'call': 'fremovexattr',
4650                         'reason': set([('fd', 'flags')])},
4651                        {'call': 'readahead',
4652                         'reason': set([('fd', 'flags')])},
4653                        {'call': 'getdents', 'reason': set([('fd', 'flags')])},
4654                        {'call': 'writev', 'reason': set([('fd', 'flags')])},
4655                        {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
4656                        {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
4657                        {'call': 'pread64', 'reason': set([('fd', 'flags')])},
4658                        {'call': 'pivot_root',
4659                         'reason': set([('dentry', 'd_inode')])},
4660                        {'call': 'signalfd4',
4661                         'reason': set([('fd', 'flags')])},
4662                        {'call': 'memfd_create',
4663                         'reason': set([('file', 'f_op')])},
4664                        {'call': 'remap_file_pages',
4665                         'reason': set([('file', 'f_op')])},
4666                        {'call': 'dup3', 'reason': set([('file', 'f_op')])},
4667                        {'call': 'read', 'reason': set([('fd', 'flags')])},
4668                        {'call': 'fchown', 'reason': set([('fd', 'flags')])},
4669                        {'call': 'mq_timedreceive',
4670                         'reason': set([('fd', 'flags')])},
4671                        {'call': 'utime', 'reason': set([('fd', 'flags')])},
4672                        {'call': 'fsync', 'reason': set([('fd', 'flags')])},
4673                        {'call': 'bpf', 'reason': set([('fd', 'flags')])},
4674                        {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
4675                        {'call': 'fsetxattr',
4676                         'reason': set([('fd', 'flags')])},
4677                        {'call': 'sendto', 'reason': set([('fd', 'flags')])},
4678                        {'call': 'mkdirat',
4679                         'reason': set([('dentry', 'd_inode')])},
```



```
4680                        {'call': 'epoll_create1',
4681                         'reason': set([('file', 'f_op')])},
4682                        {'call': 'tee', 'reason': set([('fd', 'flags')])},
4683                        {'call': 'sync_file_range',
4684                         'reason': set([('fd', 'flags')])},
4685                        {'call': 'lseek', 'reason': set([('fd', 'flags')])},
4686                        {'call': 'connect', 'reason': set([('fd', 'flags')])},
4687                        {'call': 'getsockname',
4688                         'reason': set([('fd', 'flags')])},
4689                        {'call': 'epoll_ctl',
4690                         'reason': set([('fd', 'flags'), ('file', 'f_op')])},
4691                        {'call': 'flock',
4692                         'reason': set([('fd', 'flags'), ('file', 'f_op')])},
4693                        {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
4694                        {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
4695                        {'call': 'openat', 'reason': set([('file', 'f_op')])},
4696                        {'call': 'uselib', 'reason': set([('file', 'f_op')])},
4697                        {'call': 'renameat2',
4698                         'reason': set([('dentry', 'd_inode')])},
4699                        {'call': 'accept4',
4700                         'reason': set([('fd', 'flags'), ('file', 'f_op')])},
4701                        {'call': 'old_readdir',
4702                         'reason': set([('fd', 'flags')])},
4703                        {'call': 'inotify_rm_watch',
4704                         'reason': set([('fd', 'flags')])},
4705                        {'call': 'socketpair',
4706                         'reason': set([('file', 'f_op')])},
4707                        {'call': 'utimensat',
4708                         'reason': set([('fd', 'flags')])},
4709                        {'call': 'getcwd',
4710                         'reason': set([('dentry', 'd_inode')])},
4711                        {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
4712                        {'call': 'splice', 'reason': set([('fd', 'flags')])},
4713                        {'call': 'ftruncate',
4714                         'reason': set([('dentry', 'd_inode'),
4715                                        ('fd', 'flags')])},
4716                        {'call': 'preadv', 'reason': set([('fd', 'flags')])},
4717                        {'call': 'getpeername',
4718                         'reason': set([('fd', 'flags')])},
4719                        {'call': 'shmat', 'reason': set([('file', 'f_op')])},
4720                        {'call': 'setsockopt',
4721                         'reason': set([('fd', 'flags')])},
4722                        {'call': 'mknodat',
4723                         'reason': set([('dentry', 'd_inode')])},
4724                        {'call': 'socket', 'reason': set([('file', 'f_op')])},
4725                        {'call': 'symlinkat',
4726                         'reason': set([('dentry', 'd_inode')])},
4727                        {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
4728                        {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
4729                        {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
4730                        {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
4731                        {'call': 'perf_event_open',
4732                         'reason': set([('fd', 'flags'), ('file', 'f_op')])},
4733                        {'call': 'linkat',
4734                         'reason': set([('dentry', 'd_inode')])},
4735                        {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
4736                        {'call': 'pwritev64v2',
4737                         'reason': set([('fd', 'flags')])},
4738                        {'call': 'futimesat',
4739                         'reason': set([('fd', 'flags')])},
4740                        {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
4741                        {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
4742                        {'call': 'acct', 'reason': set([('file', 'f_op')])},
4743                        {'call': 'open', 'reason': set([('file', 'f_op')])},
4744                        {'call': 'unlink',
4745                         'reason': set([('dentry', 'd_inode')])},
```

```
                      {'call': 'getsockopt',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'mq_getsetattr',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'rmdir',
                       'reason': set([('dentry', 'd_inode')])},
                      {'call': 'dup', 'reason': set([('file', 'f_op')])},
                      {'call': 'fdatasync',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'setns', 'reason': set([('file', 'f_op')])},
                      {'call': 'getdents64',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'listen', 'reason': set([('fd', 'flags')])},
                      {'call': 'copy_file_range',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'mq_timedsend',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'fgetxattr',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
                      {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                      {'call': 'swapon', 'reason': set([('file', 'f_op')])},
                      {'call': 'fallocate',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'epoll_wait',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                      {'call': 'mmap_pgoff',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'preadv64v2',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'readv', 'reason': set([('fd', 'flags')])},
                      {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                      {'call': 'fstatfs64',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'write', 'reason': set([('fd', 'flags')])},
                      {'call': 'mq_notify',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                      {'call': 'mq_open',
                       'reason': set([('dentry', 'd_inode'),
                                      ('file', 'f_op')])},
                      {'call': 'unlinkat',
                       'reason': set([('dentry', 'd_inode')])},
                      {'call': 'open_by_handle_at',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'bind', 'reason': set([('fd', 'flags')])},
                      {'call': 'flistxattr',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'sendfile64',
                       'reason': set([('fd', 'flags')])}],
 'inotify_init1': [{'call': 'keyctl',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'msgrcv',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setparam',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'ioprio_set',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getppid',
                    'reason': set([('task_struct', 'cred')])},
```



```
                   {'call': 'mq_timedreceive',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'capget',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setaffinity',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'signal',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'semtimedop',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'umount',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_rr_get_interval',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'rt_sigprocmask',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setsid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sigaltstack',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setattr',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'inotify_rm_watch',
                    'reason': set([('fsnotify_group', 'overflow_event'),
                                   ('inotify_group_private_data',
                                    'ucounts')])},
                   {'call': 'migrate_pages',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getitimer',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setpgid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'inotify_add_watch',
                    'reason': set([('fsnotify_group', 'overflow_event'),
                                   ('inotify_group_private_data',
                                    'ucounts')])},
                   {'call': 'getsid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'prlimit64',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'perf_event_open',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'rt_sigaction',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getpgid',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getpriority',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sigaction',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setns',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
                   {'call': 'get_robust_list',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'mq_timedsend',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getscheduler',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'ptrace',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getattr',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'getrusage',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_setscheduler',
```

```
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setitimer',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'ioprio_get',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'vfork',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'prctl',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'move_pages',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'setpriority',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'clone',
                    'reason': set([('task_struct', 'cred')])},
                   {'call': 'sched_getparam',
                    'reason': set([('task_struct', 'cred')])}],
 'inotify_rm_watch': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                      {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                      {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
                      {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                      {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
                      {'call': 'fremovexattr',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'readahead', 'reason': set([('fd', 'flags')])},
                      {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                      {'call': 'writev', 'reason': set([('fd', 'flags')])},
                      {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                      {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                      {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                      {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                      {'call': 'memfd_create',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'remap_file_pages',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'dup3', 'reason': set([('file', 'f_op')])},
                      {'call': 'read', 'reason': set([('fd', 'flags')])},
                      {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                      {'call': 'mq_timedreceive',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'utime', 'reason': set([('fd', 'flags')])},
                      {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                      {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                      {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                      {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                      {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                      {'call': 'epoll_create1',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'tee', 'reason': set([('fd', 'flags')])},
                      {'call': 'sync_file_range',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                      {'call': 'connect', 'reason': set([('fd', 'flags')])},
                      {'call': 'getsockname',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'epoll_ctl',
                       'reason': set([('fd', 'flags'), ('file', 'f_op')])},
                      {'call': 'flock',
                       'reason': set([('fd', 'flags'), ('file', 'f_op')])},
                      {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                      {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                      {'call': 'openat', 'reason': set([('file', 'f_op')])},
                      {'call': 'uselib', 'reason': set([('file', 'f_op')])},
                      {'call': 'accept4',
                       'reason': set([('fd', 'flags'), ('file', 'f_op')])},
                      {'call': 'old_readdir',
```



```
                       'reason': set([('fd', 'flags')])},
                      {'call': 'socketpair',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                      {'call': 'inotify_add_watch',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                      {'call': 'splice', 'reason': set([('fd', 'flags')])},
                      {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                      {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                      {'call': 'getpeername',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'shmat', 'reason': set([('file', 'f_op')])},
                      {'call': 'setsockopt',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'socket', 'reason': set([('file', 'f_op')])},
                      {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
                      {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                      {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                      {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                      {'call': 'perf_event_open',
                       'reason': set([('fd', 'flags'), ('file', 'f_op')])},
                      {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
                      {'call': 'pwritev64v2',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                      {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                      {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                      {'call': 'acct', 'reason': set([('file', 'f_op')])},
                      {'call': 'open', 'reason': set([('file', 'f_op')])},
                      {'call': 'getsockopt',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'mq_getsetattr',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'dup', 'reason': set([('file', 'f_op')])},
                      {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                      {'call': 'setns', 'reason': set([('file', 'f_op')])},
                      {'call': 'getdents64',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'listen', 'reason': set([('fd', 'flags')])},
                      {'call': 'copy_file_range',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'mq_timedsend',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                      {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
                      {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                      {'call': 'swapon', 'reason': set([('file', 'f_op')])},
                      {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                      {'call': 'epoll_wait',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                      {'call': 'mmap_pgoff',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'preadv64v2',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'readv', 'reason': set([('fd', 'flags')])},
                      {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                      {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                      {'call': 'write', 'reason': set([('fd', 'flags')])},
                      {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                      {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                      {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
                      {'call': 'open_by_handle_at',
                       'reason': set([('file', 'f_op')])},
                      {'call': 'bind', 'reason': set([('fd', 'flags')])},
```

```
5010                     {'call': 'flistxattr',
5011                      'reason': set([('fd', 'flags')])},
5012                     {'call': 'sendfile64',
5013                      'reason': set([('fd', 'flags')])}],
5014  'io_cancel': [{'call': 'swapoff',
5015                 'reason': set([('mm_struct', 'ioctx_table')])},
5016                {'call': 'remap_file_pages',
5017                 'reason': set([('mm_struct', 'ioctx_table')])},
5018                {'call': 'io_getevents',
5019                 'reason': set([('kioctx', 'user_id'),
5020                                ('kioctx_table', 'nr'),
5021                                ('mm_struct', 'ioctx_table')])},
5022                {'call': 'migrate_pages',
5023                 'reason': set([('mm_struct', 'ioctx_table')])},
5024                {'call': 'shmdt',
5025                 'reason': set([('mm_struct', 'ioctx_table')])},
5026                {'call': 'brk', 'reason': set([('mm_struct', 'ioctx_table')])},
5027                {'call': 'get_mempolicy',
5028                 'reason': set([('mm_struct', 'ioctx_table')])},
5029                {'call': 'io_submit', 'reason': set([('kioctx', 'user_id')])},
5030                {'call': 'getrusage',
5031                 'reason': set([('mm_struct', 'ioctx_table')])},
5032                {'call': 'io_setup',
5033                 'reason': set([('kioctx', 'user_id'),
5034                                ('kioctx_table', 'nr'),
5035                                ('mm_struct', 'ioctx_table')])},
5036                {'call': 'mremap',
5037                 'reason': set([('mm_struct', 'ioctx_table')])},
5038                {'call': 'io_destroy',
5039                 'reason': set([('kioctx', 'user_id'),
5040                                ('kioctx_table', 'nr'),
5041                                ('mm_struct', 'ioctx_table')])},
5042                {'call': 'mbind',
5043                 'reason': set([('mm_struct', 'ioctx_table')])},
5044                {'call': 'prctl',
5045                 'reason': set([('mm_struct', 'ioctx_table')])},
5046                {'call': 'move_pages',
5047                 'reason': set([('mm_struct', 'ioctx_table')])},
5048                {'call': 'modify_ldt',
5049                 'reason': set([('mm_struct', 'ioctx_table')])},
5050                {'call': 'mincore',
5051                 'reason': set([('mm_struct', 'ioctx_table')])}],
5052  'io_destroy': [{'call': 'swapoff',
5053                  'reason': set([('mm_struct', 'ioctx_table')])},
5054                 {'call': 'remap_file_pages',
5055                  'reason': set([('mm_struct', 'ioctx_table')])},
5056                 {'call': 'io_getevents',
5057                  'reason': set([('kioctx', 'max_reqs'),
5058                                 ('kioctx', 'mmap_base'),
5059                                 ('kioctx', 'mmap_size'),
5060                                 ('kioctx', 'user_id'),
5061                                 ('kioctx_table', 'nr'),
5062                                 ('mm_struct', 'ioctx_table')])},
5063                 {'call': 'migrate_pages',
5064                  'reason': set([('mm_struct', 'ioctx_table')])},
5065                 {'call': 'shmdt',
5066                  'reason': set([('mm_struct', 'ioctx_table')])},
5067                 {'call': 'brk',
5068                  'reason': set([('mm_struct', 'ioctx_table')])},
5069                 {'call': 'get_mempolicy',
5070                  'reason': set([('mm_struct', 'ioctx_table')])},
5071                 {'call': 'io_submit',
5072                  'reason': set([('kioctx', 'max_reqs'),
5073                                 ('kioctx', 'mmap_base'),
5074                                 ('kioctx', 'mmap_size'),
5075                                 ('kioctx', 'user_id')])},
```



```
5076                {'call': 'getrusage',
5077                 'reason': set([('mm_struct', 'ioctx_table')])},
5078                {'call': 'io_setup',
5079                 'reason': set([('kioctx', 'max_reqs'),
5080                                ('kioctx', 'mmap_base'),
5081                                ('kioctx', 'mmap_size'),
5082                                ('kioctx', 'user_id'),
5083                                ('kioctx_table', 'nr'),
5084                                ('mm_struct', 'ioctx_table')])},
5085                {'call': 'mremap',
5086                 'reason': set([('mm_struct', 'ioctx_table')])},
5087                {'call': 'mbind',
5088                 'reason': set([('mm_struct', 'ioctx_table')])},
5089                {'call': 'prctl',
5090                 'reason': set([('mm_struct', 'ioctx_table')])},
5091                {'call': 'move_pages',
5092                 'reason': set([('mm_struct', 'ioctx_table')])},
5093                {'call': 'modify_ldt',
5094                 'reason': set([('mm_struct', 'ioctx_table')])},
5095                {'call': 'mincore',
5096                 'reason': set([('mm_struct', 'ioctx_table')])},
5097                {'call': 'io_cancel',
5098                 'reason': set([('kioctx', 'max_reqs'),
5099                                ('kioctx', 'mmap_base'),
5100                                ('kioctx', 'mmap_size'),
5101                                ('kioctx', 'user_id'),
5102                                ('kioctx_table', 'nr'),
5103                                ('mm_struct', 'ioctx_table')])}],
5104  'io_getevents': [{'call': 'keyctl',
5105                    'reason': set([('task_struct', 'timer_slack_ns')])},
5106                   {'call': 'rt_sigtimedwait',
5107                    'reason': set([('task_struct', 'timer_slack_ns')])},
5108                   {'call': 'msgrcv',
5109                    'reason': set([('task_struct', 'timer_slack_ns')])},
5110                   {'call': 'kill',
5111                    'reason': set([('task_struct', 'timer_slack_ns')])},
5112                   {'call': 'swapoff',
5113                    'reason': set([('mm_struct', 'ioctx_table')])},
5114                   {'call': 'sched_getaffinity',
5115                    'reason': set([('task_struct', 'timer_slack_ns')])},
5116                   {'call': 'sched_setparam',
5117                    'reason': set([('task_struct', 'timer_slack_ns')])},
5118                   {'call': 'ioprio_set',
5119                    'reason': set([('task_struct', 'timer_slack_ns')])},
5120                   {'call': 'remap_file_pages',
5121                    'reason': set([('mm_struct', 'ioctx_table')])},
5122                   {'call': 'getppid',
5123                    'reason': set([('task_struct', 'timer_slack_ns')])},
5124                   {'call': 'mq_timedreceive',
5125                    'reason': set([('task_struct', 'timer_slack_ns')])},
5126                   {'call': 'capget',
5127                    'reason': set([('task_struct', 'timer_slack_ns')])},
5128                   {'call': 'sched_setaffinity',
5129                    'reason': set([('task_struct', 'timer_slack_ns')])},
5130                   {'call': 'signal',
5131                    'reason': set([('task_struct', 'timer_slack_ns')])},
5132                   {'call': 'semtimedop',
5133                    'reason': set([('task_struct', 'timer_slack_ns')])},
5134                   {'call': 'umount',
5135                    'reason': set([('task_struct', 'timer_slack_ns')])},
5136                   {'call': 'sched_rr_get_interval',
5137                    'reason': set([('task_struct', 'timer_slack_ns')])},
5138                   {'call': 'rt_sigprocmask',
5139                    'reason': set([('task_struct', 'timer_slack_ns')])},
5140                   {'call': 'setsid',
5141                    'reason': set([('task_struct', 'timer_slack_ns')])},
```

```
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'migrate_pages',
                 'reason': set([('mm_struct', 'ioctx_table'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'shmdt',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'brk',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'get_mempolicy',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'io_submit',
                 'reason': set([('kioctx', 'user_id')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'getrusage',
                 'reason': set([('mm_struct', 'ioctx_table'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'io_setup',
                 'reason': set([('kioctx', 'user_id'),
                                ('kioctx_table', 'nr'),
                                ('mm_struct', 'ioctx_table')])},
                {'call': 'mremap',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'io_destroy',
                 'reason': set([('kioctx', 'user_id'),
```



```
                                ('kioctx_table', 'nr'),
                                ('mm_struct', 'ioctx_table')])},
                {'call': 'mbind',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'prctl',
                 'reason': set([('mm_struct', 'ioctx_table'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'move_pages',
                 'reason': set([('mm_struct', 'ioctx_table'),
                                ('task_struct', 'timer_slack_ns')])},
                {'call': 'modify_ldt',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'mincore',
                 'reason': set([('mm_struct', 'ioctx_table')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'timer_slack_ns')])},
                {'call': 'io_cancel',
                 'reason': set([('kioctx', 'user_id'),
                                ('kioctx_table', 'nr'),
                                ('mm_struct', 'ioctx_table')])}],
 'io_setup': [{'call': 'io_getevents',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_submit',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_destroy',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_cancel',
               'reason': set([('kioctx', 'cpu'),
                              ('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])}],
 'io_submit': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setparam',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioprio_set',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedreceive',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
```

```
5274                {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
5275                {'call': 'semtimedop',
5276                 'reason': set([('mm_segment_t', 'seg')])},
5277                {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
5278                {'call': 'sched_rr_get_interval',
5279                 'reason': set([('mm_segment_t', 'seg')])},
5280                {'call': 'rt_sigprocmask',
5281                 'reason': set([('mm_segment_t', 'seg')])},
5282                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
5283                {'call': 'sigaltstack',
5284                 'reason': set([('mm_segment_t', 'seg')])},
5285                {'call': 'sched_setattr',
5286                 'reason': set([('mm_segment_t', 'seg')])},
5287                {'call': 'migrate_pages',
5288                 'reason': set([('mm_segment_t', 'seg')])},
5289                {'call': 'getitimer',
5290                 'reason': set([('mm_segment_t', 'seg')])},
5291                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
5292                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
5293                {'call': 'prlimit64',
5294                 'reason': set([('mm_segment_t', 'seg')])},
5295                {'call': 'perf_event_open',
5296                 'reason': set([('mm_segment_t', 'seg')])},
5297                {'call': 'rt_sigaction',
5298                 'reason': set([('mm_segment_t', 'seg')])},
5299                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
5300                {'call': 'getpriority',
5301                 'reason': set([('mm_segment_t', 'seg')])},
5302                {'call': 'sigaction',
5303                 'reason': set([('mm_segment_t', 'seg')])},
5304                {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
5305                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
5306                {'call': 'get_robust_list',
5307                 'reason': set([('mm_segment_t', 'seg')])},
5308                {'call': 'mq_timedsend',
5309                 'reason': set([('mm_segment_t', 'seg')])},
5310                {'call': 'sched_getscheduler',
5311                 'reason': set([('mm_segment_t', 'seg')])},
5312                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
5313                {'call': 'sched_getattr',
5314                 'reason': set([('mm_segment_t', 'seg')])},
5315                {'call': 'getrusage',
5316                 'reason': set([('mm_segment_t', 'seg')])},
5317                {'call': 'sched_setscheduler',
5318                 'reason': set([('mm_segment_t', 'seg')])},
5319                {'call': 'setitimer',
5320                 'reason': set([('mm_segment_t', 'seg')])},
5321                {'call': 'ioprio_get',
5322                 'reason': set([('mm_segment_t', 'seg')])},
5323                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
5324                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
5325                {'call': 'move_pages',
5326                 'reason': set([('mm_segment_t', 'seg')])},
5327                {'call': 'setpriority',
5328                 'reason': set([('mm_segment_t', 'seg')])},
5329                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
5330                {'call': 'sched_getparam',
5331                 'reason': set([('mm_segment_t', 'seg')])}],
5332  'ioctl': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
5333            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
5334            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
5335            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
5336            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
5337            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
5338            {'call': 'writev', 'reason': set([('fd', 'flags')])},
5339            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
```



```
5340            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
5341            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
5342            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
5343            {'call': 'read', 'reason': set([('fd', 'flags')])},
5344            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
5345            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
5346            {'call': 'utime', 'reason': set([('fd', 'flags')])},
5347            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
5348            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
5349            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
5350            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
5351            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
5352            {'call': 'tee', 'reason': set([('fd', 'flags')])},
5353            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
5354            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
5355            {'call': 'connect', 'reason': set([('fd', 'flags')])},
5356            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
5357            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
5358            {'call': 'flock', 'reason': set([('fd', 'flags')])},
5359            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
5360            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
5361            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
5362            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
5363            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
5364            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
5365            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
5366            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
5367            {'call': 'splice', 'reason': set([('fd', 'flags')])},
5368            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
5369            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
5370            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
5371            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
5372            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
5373            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
5374            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
5375            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
5376            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
5377            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
5378            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
5379            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
5380            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
5381            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
5382            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
5383            {'call': 'listen', 'reason': set([('fd', 'flags')])},
5384            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
5385            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
5386            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
5387            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
5388            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
5389            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
5390            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
5391            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
5392            {'call': 'readv', 'reason': set([('fd', 'flags')])},
5393            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
5394            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
5395            {'call': 'write', 'reason': set([('fd', 'flags')])},
5396            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
5397            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
5398            {'call': 'bind', 'reason': set([('fd', 'flags')])},
5399            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
5400            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
5401  'ioperm': [{'call': 'keyctl',
5402              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5403             {'call': 'rt_sigtimedwait',
5404              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5405             {'call': 'msgrcv',
```

```
5406              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5407             {'call': 'kill',
5408              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5409             {'call': 'sched_getaffinity',
5410              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5411             {'call': 'sched_setparam',
5412              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5413             {'call': 'ioprio_set',
5414              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5415             {'call': 'getppid',
5416              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5417             {'call': 'mq_timedreceive',
5418              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5419             {'call': 'capget',
5420              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5421             {'call': 'sched_setaffinity',
5422              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5423             {'call': 'signal',
5424              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5425             {'call': 'semtimedop',
5426              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5427             {'call': 'umount',
5428              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5429             {'call': 'sched_rr_get_interval',
5430              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5431             {'call': 'rt_sigprocmask',
5432              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5433             {'call': 'setsid',
5434              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5435             {'call': 'sigaltstack',
5436              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5437             {'call': 'sched_setattr',
5438              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5439             {'call': 'migrate_pages',
5440              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5441             {'call': 'getitimer',
5442              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5443             {'call': 'setpgid',
5444              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5445             {'call': 'getsid',
5446              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5447             {'call': 'prlimit64',
5448              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5449             {'call': 'perf_event_open',
5450              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5451             {'call': 'rt_sigaction',
5452              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5453             {'call': 'getpgid',
5454              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5455             {'call': 'getpriority',
5456              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5457             {'call': 'sigaction',
5458              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5459             {'call': 'setns',
5460              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5461             {'call': 'fork',
5462              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5463             {'call': 'get_robust_list',
5464              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5465             {'call': 'mq_timedsend',
5466              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5467             {'call': 'sched_getscheduler',
5468              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5469             {'call': 'ptrace',
5470              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5471             {'call': 'sched_getattr',
```



```
5472              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5473             {'call': 'getrusage',
5474              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5475             {'call': 'sched_setscheduler',
5476              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5477             {'call': 'setitimer',
5478              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5479             {'call': 'ioprio_get',
5480              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5481             {'call': 'vfork',
5482              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5483             {'call': 'prctl',
5484              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5485             {'call': 'move_pages',
5486              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5487             {'call': 'setpriority',
5488              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5489             {'call': 'clone',
5490              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
5491             {'call': 'sched_getparam',
5492              'reason': set([('thread_struct', 'io_bitmap_ptr')])}],
5493  'ioprio_get': [{'call': 'keyctl',
5494                  'reason': set([('task_struct', 'cred'),
5495                                 ('task_struct', 'io_context'),
5496                                 ('task_struct', 'real_cred')])},
5497                 {'call': 'rt_sigtimedwait',
5498                  'reason': set([('task_struct', 'cred'),
5499                                 ('task_struct', 'io_context'),
5500                                 ('task_struct', 'real_cred')])},
5501                 {'call': 'msgrcv',
5502                  'reason': set([('task_struct', 'cred'),
5503                                 ('task_struct', 'io_context'),
5504                                 ('task_struct', 'real_cred')])},
5505                 {'call': 'kill',
5506                  'reason': set([('task_struct', 'cred'),
5507                                 ('task_struct', 'io_context'),
5508                                 ('task_struct', 'real_cred')])},
5509                 {'call': 'sched_getaffinity',
5510                  'reason': set([('task_struct', 'cred'),
5511                                 ('task_struct', 'io_context'),
5512                                 ('task_struct', 'real_cred')])},
5513                 {'call': 'sched_setparam',
5514                  'reason': set([('task_struct', 'cred'),
5515                                 ('task_struct', 'io_context'),
5516                                 ('task_struct', 'real_cred')])},
5517                 {'call': 'ioprio_set',
5518                  'reason': set([('task_struct', 'cred'),
5519                                 ('task_struct', 'io_context'),
5520                                 ('task_struct', 'real_cred')])},
5521                 {'call': 'getppid',
5522                  'reason': set([('task_struct', 'cred'),
5523                                 ('task_struct', 'io_context'),
5524                                 ('task_struct', 'real_cred')])},
5525                 {'call': 'mq_timedreceive',
5526                  'reason': set([('task_struct', 'cred'),
5527                                 ('task_struct', 'io_context'),
5528                                 ('task_struct', 'real_cred')])},
5529                 {'call': 'capget',
5530                  'reason': set([('task_struct', 'cred'),
5531                                 ('task_struct', 'io_context'),
5532                                 ('task_struct', 'real_cred')])},
5533                 {'call': 'sched_setaffinity',
5534                  'reason': set([('task_struct', 'cred'),
5535                                 ('task_struct', 'io_context'),
5536                                 ('task_struct', 'real_cred')])},
5537                 {'call': 'signal',
```

```
5538                  'reason': set([('task_struct', 'cred'),
5539                                 ('task_struct', 'io_context'),
5540                                 ('task_struct', 'real_cred')])},
5541                {'call': 'semtimedop',
5542                 'reason': set([('task_struct', 'cred'),
5543                                ('task_struct', 'io_context'),
5544                                ('task_struct', 'real_cred')])},
5545                {'call': 'umount',
5546                 'reason': set([('task_struct', 'cred'),
5547                                ('task_struct', 'io_context'),
5548                                ('task_struct', 'real_cred')])},
5549                {'call': 'sched_rr_get_interval',
5550                 'reason': set([('task_struct', 'cred'),
5551                                ('task_struct', 'io_context'),
5552                                ('task_struct', 'real_cred')])},
5553                {'call': 'rt_sigprocmask',
5554                 'reason': set([('task_struct', 'cred'),
5555                                ('task_struct', 'io_context'),
5556                                ('task_struct', 'real_cred')])},
5557                {'call': 'setsid',
5558                 'reason': set([('task_struct', 'cred'),
5559                                ('task_struct', 'io_context'),
5560                                ('task_struct', 'real_cred')])},
5561                {'call': 'sigaltstack',
5562                 'reason': set([('task_struct', 'cred'),
5563                                ('task_struct', 'io_context'),
5564                                ('task_struct', 'real_cred')])},
5565                {'call': 'sched_setattr',
5566                 'reason': set([('task_struct', 'cred'),
5567                                ('task_struct', 'io_context'),
5568                                ('task_struct', 'real_cred')])},
5569                {'call': 'migrate_pages',
5570                 'reason': set([('task_struct', 'cred'),
5571                                ('task_struct', 'io_context'),
5572                                ('task_struct', 'real_cred')])},
5573                {'call': 'getitimer',
5574                 'reason': set([('task_struct', 'cred'),
5575                                ('task_struct', 'io_context'),
5576                                ('task_struct', 'real_cred')])},
5577                {'call': 'setpgid',
5578                 'reason': set([('task_struct', 'cred'),
5579                                ('task_struct', 'io_context'),
5580                                ('task_struct', 'real_cred')])},
5581                {'call': 'getsid',
5582                 'reason': set([('task_struct', 'cred'),
5583                                ('task_struct', 'io_context'),
5584                                ('task_struct', 'real_cred')])},
5585                {'call': 'prlimit64',
5586                 'reason': set([('task_struct', 'cred'),
5587                                ('task_struct', 'io_context'),
5588                                ('task_struct', 'real_cred')])},
5589                {'call': 'perf_event_open',
5590                 'reason': set([('task_struct', 'cred'),
5591                                ('task_struct', 'io_context'),
5592                                ('task_struct', 'real_cred')])},
5593                {'call': 'rt_sigaction',
5594                 'reason': set([('task_struct', 'cred'),
5595                                ('task_struct', 'io_context'),
5596                                ('task_struct', 'real_cred')])},
5597                {'call': 'getpgid',
5598                 'reason': set([('task_struct', 'cred'),
5599                                ('task_struct', 'io_context'),
5600                                ('task_struct', 'real_cred')])},
5601                {'call': 'getpriority',
5602                 'reason': set([('task_struct', 'cred'),
5603                                ('task_struct', 'io_context'),
```



```
5604                                ('task_struct', 'real_cred')])},
5605                {'call': 'sigaction',
5606                 'reason': set([('task_struct', 'cred'),
5607                                ('task_struct', 'io_context'),
5608                                ('task_struct', 'real_cred')])},
5609                {'call': 'setns',
5610                 'reason': set([('task_struct', 'cred'),
5611                                ('task_struct', 'io_context'),
5612                                ('task_struct', 'real_cred')])},
5613                {'call': 'fork',
5614                 'reason': set([('task_struct', 'cred'),
5615                                ('task_struct', 'io_context'),
5616                                ('task_struct', 'real_cred')])},
5617                {'call': 'get_robust_list',
5618                 'reason': set([('task_struct', 'cred'),
5619                                ('task_struct', 'io_context'),
5620                                ('task_struct', 'real_cred')])},
5621                {'call': 'mq_timedsend',
5622                 'reason': set([('task_struct', 'cred'),
5623                                ('task_struct', 'io_context'),
5624                                ('task_struct', 'real_cred')])},
5625                {'call': 'sched_getscheduler',
5626                 'reason': set([('task_struct', 'cred'),
5627                                ('task_struct', 'io_context'),
5628                                ('task_struct', 'real_cred')])},
5629                {'call': 'ptrace',
5630                 'reason': set([('task_struct', 'cred'),
5631                                ('task_struct', 'io_context'),
5632                                ('task_struct', 'real_cred')])},
5633                {'call': 'sched_getattr',
5634                 'reason': set([('task_struct', 'cred'),
5635                                ('task_struct', 'io_context'),
5636                                ('task_struct', 'real_cred')])},
5637                {'call': 'getrusage',
5638                 'reason': set([('task_struct', 'cred'),
5639                                ('task_struct', 'io_context'),
5640                                ('task_struct', 'real_cred')])},
5641                {'call': 'sched_setscheduler',
5642                 'reason': set([('task_struct', 'cred'),
5643                                ('task_struct', 'io_context'),
5644                                ('task_struct', 'real_cred')])},
5645                {'call': 'setitimer',
5646                 'reason': set([('task_struct', 'cred'),
5647                                ('task_struct', 'io_context'),
5648                                ('task_struct', 'real_cred')])},
5649                {'call': 'vfork',
5650                 'reason': set([('task_struct', 'cred'),
5651                                ('task_struct', 'io_context'),
5652                                ('task_struct', 'real_cred')])},
5653                {'call': 'prctl',
5654                 'reason': set([('task_struct', 'cred'),
5655                                ('task_struct', 'io_context'),
5656                                ('task_struct', 'real_cred')])},
5657                {'call': 'move_pages',
5658                 'reason': set([('task_struct', 'cred'),
5659                                ('task_struct', 'io_context'),
5660                                ('task_struct', 'real_cred')])},
5661                {'call': 'setpriority',
5662                 'reason': set([('task_struct', 'cred'),
5663                                ('task_struct', 'io_context'),
5664                                ('task_struct', 'real_cred')])},
5665                {'call': 'clone',
5666                 'reason': set([('task_struct', 'cred'),
5667                                ('task_struct', 'io_context'),
5668                                ('task_struct', 'real_cred')])},
5669                {'call': 'sched_getparam',
```

```
                'reason': set([('task_struct', 'cred'),
                               ('task_struct', 'io_context'),
                               ('task_struct', 'real_cred')])}],
 'ioprio_set': [{'call': 'keyctl',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigtimedwait',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'msgrcv',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'kill',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getaffinity',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setparam',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'capget',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'signal',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'umount',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
```



```
                {'call': 'getsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
```

```
5802                {'call': 'clone',
5803                 'reason': set([('task_struct', 'cred'),
5804                                ('task_struct', 'real_cred')])},
5805                {'call': 'sched_getparam',
5806                 'reason': set([('task_struct', 'cred'),
5807                                ('task_struct', 'real_cred')])}],
5808  'keyctl': [{'call': 'rt_sigtimedwait',
5809              'reason': set([('task_struct', 'cred'),
5810                             ('task_struct', 'mm'),
5811                             ('task_struct', 'pid'),
5812                             ('task_struct', 'real_cred')])},
5813             {'call': 'setfsuid',
5814              'reason': set([('cred', 'session_keyring')])},
5815             {'call': 'msgrcv',
5816              'reason': set([('task_struct', 'cred'),
5817                             ('task_struct', 'mm'),
5818                             ('task_struct', 'pid'),
5819                             ('task_struct', 'real_cred')])},
5820             {'call': 'kill',
5821              'reason': set([('task_struct', 'cred'),
5822                             ('task_struct', 'mm'),
5823                             ('task_struct', 'pid'),
5824                             ('task_struct', 'real_cred')])},
5825             {'call': 'getresuid16',
5826              'reason': set([('cred', 'session_keyring')])},
5827             {'call': 'getresgid',
5828              'reason': set([('cred', 'session_keyring')])},
5829             {'call': 'sched_getaffinity',
5830              'reason': set([('task_struct', 'cred'),
5831                             ('task_struct', 'mm'),
5832                             ('task_struct', 'pid'),
5833                             ('task_struct', 'real_cred')])},
5834             {'call': 'sched_setparam',
5835              'reason': set([('task_struct', 'cred'),
5836                             ('task_struct', 'mm'),
5837                             ('task_struct', 'pid'),
5838                             ('task_struct', 'real_cred')])},
5839             {'call': 'setgid',
5840              'reason': set([('cred', 'egid'),
5841                             ('cred', 'gid'),
5842                             ('cred', 'session_keyring'),
5843                             ('cred', 'sgid')])},
5844             {'call': 'ioprio_set',
5845              'reason': set([('cred', 'session_keyring'),
5846                             ('task_struct', 'cred'),
5847                             ('task_struct', 'mm'),
5848                             ('task_struct', 'pid'),
5849                             ('task_struct', 'real_cred')])},
5850             {'call': 'capset', 'reason': set([('cred', 'session_keyring')])},
5851             {'call': 'getppid',
5852              'reason': set([('task_struct', 'cred'),
5853                             ('task_struct', 'mm'),
5854                             ('task_struct', 'pid'),
5855                             ('task_struct', 'real_cred')])},
5856             {'call': 'mq_timedreceive',
5857              'reason': set([('task_struct', 'cred'),
5858                             ('task_struct', 'mm'),
5859                             ('task_struct', 'pid'),
5860                             ('task_struct', 'real_cred')])},
5861             {'call': 'getresgid16',
5862              'reason': set([('cred', 'session_keyring')])},
5863             {'call': 'capget',
5864              'reason': set([('task_struct', 'cred'),
5865                             ('task_struct', 'mm'),
5866                             ('task_struct', 'pid'),
5867                             ('task_struct', 'real_cred')])},
```



```
5868             {'call': 'sched_setaffinity',
5869              'reason': set([('cred', 'session_keyring'),
5870                             ('task_struct', 'cred'),
5871                             ('task_struct', 'mm'),
5872                             ('task_struct', 'pid'),
5873                             ('task_struct', 'real_cred')])},
5874             {'call': 'setfsgid',
5875              'reason': set([('cred', 'session_keyring')])},
5876             {'call': 'unshare', 'reason': set([('cred', 'session_keyring')])},
5877             {'call': 'signal',
5878              'reason': set([('task_struct', 'cred'),
5879                             ('task_struct', 'mm'),
5880                             ('task_struct', 'pid'),
5881                             ('task_struct', 'real_cred')])},
5882             {'call': 'setreuid',
5883              'reason': set([('cred', 'euid'),
5884                             ('cred', 'session_keyring'),
5885                             ('cred', 'suid'),
5886                             ('cred', 'uid')])},
5887             {'call': 'semtimedop',
5888              'reason': set([('task_struct', 'cred'),
5889                             ('task_struct', 'mm'),
5890                             ('task_struct', 'pid'),
5891                             ('task_struct', 'real_cred')])},
5892             {'call': 'umount',
5893              'reason': set([('task_struct', 'cred'),
5894                             ('task_struct', 'mm'),
5895                             ('task_struct', 'pid'),
5896                             ('task_struct', 'real_cred')])},
5897             {'call': 'sched_rr_get_interval',
5898              'reason': set([('task_struct', 'cred'),
5899                             ('task_struct', 'mm'),
5900                             ('task_struct', 'pid'),
5901                             ('task_struct', 'real_cred')])},
5902             {'call': 'epoll_create1',
5903              'reason': set([('cred', 'session_keyring')])},
5904             {'call': 'getresuid',
5905              'reason': set([('cred', 'session_keyring')])},
5906             {'call': 'rt_sigprocmask',
5907              'reason': set([('task_struct', 'cred'),
5908                             ('task_struct', 'mm'),
5909                             ('task_struct', 'pid'),
5910                             ('task_struct', 'real_cred')])},
5911             {'call': 'setsid',
5912              'reason': set([('task_struct', 'cred'),
5913                             ('task_struct', 'mm'),
5914                             ('task_struct', 'pid'),
5915                             ('task_struct', 'real_cred')])},
5916             {'call': 'sigaltstack',
5917              'reason': set([('task_struct', 'cred'),
5918                             ('task_struct', 'mm'),
5919                             ('task_struct', 'pid'),
5920                             ('task_struct', 'real_cred')])},
5921             {'call': 'sched_setattr',
5922              'reason': set([('task_struct', 'cred'),
5923                             ('task_struct', 'mm'),
5924                             ('task_struct', 'pid'),
5925                             ('task_struct', 'real_cred')])},
5926             {'call': 'migrate_pages',
5927              'reason': set([('cred', 'session_keyring'),
5928                             ('task_struct', 'cred'),
5929                             ('task_struct', 'mm'),
5930                             ('task_struct', 'pid'),
5931                             ('task_struct', 'real_cred')])},
5932             {'call': 'getitimer',
5933              'reason': set([('task_struct', 'cred'),
```

```
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'setpgid',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'setresgid',
             'reason': set([('cred', 'egid'),
                            ('cred', 'gid'),
                            ('cred', 'session_keyring'),
                            ('cred', 'sgid')])},
            {'call': 'setregid',
             'reason': set([('cred', 'egid'),
                            ('cred', 'gid'),
                            ('cred', 'session_keyring'),
                            ('cred', 'sgid')])},
            {'call': 'getsid',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'prlimit64',
             'reason': set([('cred', 'session_keyring'),
                            ('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'perf_event_open',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'getgroups16',
             'reason': set([('cred', 'session_keyring')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'request_key',
             'reason': set([('key', 'description'),
                            ('key', 'perm'),
                            ('key', 'quotalen'),
                            ('key', 'serial'),
                            ('key_type', 'name'),
                            ('key_type', 'read')])},
            {'call': 'getpgid',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'getpriority',
             'reason': set([('cred', 'session_keyring'),
                            ('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'sigaction',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'faccessat',
             'reason': set([('cred', 'session_keyring')])},
```



```
            {'call': 'setns',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'fork',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'ptrace',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'getrusage',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'setresuid',
             'reason': set([('cred', 'euid'),
                            ('cred', 'session_keyring'),
                            ('cred', 'suid'),
                            ('cred', 'uid')])},
            {'call': 'setitimer',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'ioprio_get',
             'reason': set([('cred', 'session_keyring'),
                            ('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
            {'call': 'vfork',
             'reason': set([('task_struct', 'cred'),
                            ('task_struct', 'mm'),
                            ('task_struct', 'pid'),
                            ('task_struct', 'real_cred')])},
```

```
6066            {'call': 'setuid',
6067             'reason': set([('cred', 'euid'),
6068                            ('cred', 'session_keyring'),
6069                            ('cred', 'suid'),
6070                            ('cred', 'uid')])},
6071            {'call': 'prctl',
6072             'reason': set([('task_struct', 'cred'),
6073                            ('task_struct', 'mm'),
6074                            ('task_struct', 'pid'),
6075                            ('task_struct', 'real_cred')])},
6076            {'call': 'move_pages',
6077             'reason': set([('task_struct', 'cred'),
6078                            ('task_struct', 'mm'),
6079                            ('task_struct', 'pid'),
6080                            ('task_struct', 'real_cred')])},
6081            {'call': 'getgroups',
6082             'reason': set([('cred', 'session_keyring')])},
6083            {'call': 'setpriority',
6084             'reason': set([('cred', 'session_keyring'),
6085                            ('task_struct', 'cred'),
6086                            ('task_struct', 'mm'),
6087                            ('task_struct', 'pid'),
6088                            ('task_struct', 'real_cred')])},
6089            {'call': 'clone',
6090             'reason': set([('task_struct', 'cred'),
6091                            ('task_struct', 'mm'),
6092                            ('task_struct', 'pid'),
6093                            ('task_struct', 'real_cred')])},
6094            {'call': 'sched_getparam',
6095             'reason': set([('task_struct', 'cred'),
6096                            ('task_struct', 'mm'),
6097                            ('task_struct', 'pid'),
6098                            ('task_struct', 'real_cred')])}],
6099  'kill': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
6100           {'call': 'rt_sigtimedwait',
6101            'reason': set([('task_struct', 'cred')])},
6102           {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
6103           {'call': 'sched_getaffinity',
6104            'reason': set([('task_struct', 'cred')])},
6105           {'call': 'sched_setparam',
6106            'reason': set([('task_struct', 'cred')])},
6107           {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
6108           {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
6109           {'call': 'mq_timedreceive',
6110            'reason': set([('task_struct', 'cred')])},
6111           {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
6112           {'call': 'sched_setaffinity',
6113            'reason': set([('task_struct', 'cred')])},
6114           {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
6115           {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
6116           {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
6117           {'call': 'sched_rr_get_interval',
6118            'reason': set([('task_struct', 'cred')])},
6119           {'call': 'rt_sigprocmask',
6120            'reason': set([('task_struct', 'cred')])},
6121           {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
6122           {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
6123           {'call': 'sched_setattr', 'reason': set([('task_struct', 'cred')])},
6124           {'call': 'migrate_pages', 'reason': set([('task_struct', 'cred')])},
6125           {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
6126           {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
6127           {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
6128           {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
6129           {'call': 'perf_event_open',
6130            'reason': set([('task_struct', 'cred')])},
6131           {'call': 'rt_sigaction', 'reason': set([('task_struct', 'cred')])},
```



```
6132           {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
6133           {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
6134           {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
6135           {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
6136           {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
6137           {'call': 'get_robust_list',
6138            'reason': set([('task_struct', 'cred')])},
6139           {'call': 'mq_timedsend', 'reason': set([('task_struct', 'cred')])},
6140           {'call': 'sched_getscheduler',
6141            'reason': set([('task_struct', 'cred')])},
6142           {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
6143           {'call': 'sched_getattr', 'reason': set([('task_struct', 'cred')])},
6144           {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
6145           {'call': 'sched_setscheduler',
6146            'reason': set([('task_struct', 'cred')])},
6147           {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
6148           {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
6149           {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
6150           {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
6151           {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
6152           {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
6153           {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
6154           {'call': 'sched_getparam',
6155            'reason': set([('task_struct', 'cred')])}],
6156  'lgetxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
6157                {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
6158                {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
6159                {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
6160                {'call': 'remap_file_pages',
6161                 'reason': set([('path', 'dentry')])},
6162                {'call': 'dup3', 'reason': set([('path', 'dentry')])},
6163                {'call': 'unshare', 'reason': set([('path', 'dentry')])},
6164                {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
6165                {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
6166                {'call': 'flock', 'reason': set([('path', 'dentry')])},
6167                {'call': 'openat', 'reason': set([('path', 'dentry')])},
6168                {'call': 'lookup_dcookie',
6169                 'reason': set([('path', 'dentry')])},
6170                {'call': 'uselib', 'reason': set([('path', 'dentry')])},
6171                {'call': 'accept4', 'reason': set([('path', 'dentry')])},
6172                {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
6173                {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
6174                {'call': 'shmat', 'reason': set([('path', 'dentry')])},
6175                {'call': 'socket', 'reason': set([('path', 'dentry')])},
6176                {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
6177                {'call': 'perf_event_open',
6178                 'reason': set([('path', 'dentry')])},
6179                {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
6180                {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
6181                {'call': 'acct', 'reason': set([('path', 'dentry')])},
6182                {'call': 'open', 'reason': set([('path', 'dentry')])},
6183                {'call': 'dup', 'reason': set([('path', 'dentry')])},
6184                {'call': 'setns', 'reason': set([('path', 'dentry')])},
6185                {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
6186                {'call': 'swapon', 'reason': set([('path', 'dentry')])},
6187                {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
6188                {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
6189                {'call': 'open_by_handle_at',
6190                 'reason': set([('path', 'dentry')])}],
6191  'linkat': [{'call': 'eventfd2', 'reason': set([('path', 'mnt')])},
6192             {'call': 'swapoff', 'reason': set([('path', 'mnt')])},
6193             {'call': 'pivot_root', 'reason': set([('path', 'mnt')])},
6194             {'call': 'memfd_create', 'reason': set([('path', 'mnt')])},
6195             {'call': 'remap_file_pages', 'reason': set([('path', 'mnt')])},
6196             {'call': 'dup3', 'reason': set([('path', 'mnt')])},
6197             {'call': 'unshare', 'reason': set([('path', 'mnt')])},
```

```
            {'call': 'epoll_create1', 'reason': set([('path', 'mnt')])},
            {'call': 'epoll_ctl', 'reason': set([('path', 'mnt')])},
            {'call': 'flock', 'reason': set([('path', 'mnt')])},
            {'call': 'openat', 'reason': set([('path', 'mnt')])},
            {'call': 'lookup_dcookie', 'reason': set([('path', 'mnt')])},
            {'call': 'uselib', 'reason': set([('path', 'mnt')])},
            {'call': 'accept4', 'reason': set([('path', 'mnt')])},
            {'call': 'socketpair', 'reason': set([('path', 'mnt')])},
            {'call': 'getcwd', 'reason': set([('path', 'mnt')])},
            {'call': 'shmat', 'reason': set([('path', 'mnt')])},
            {'call': 'socket', 'reason': set([('path', 'mnt')])},
            {'call': 'pipe2', 'reason': set([('path', 'mnt')])},
            {'call': 'perf_event_open', 'reason': set([('path', 'mnt')])},
            {'call': 'shmdt', 'reason': set([('path', 'mnt')])},
            {'call': 'quotactl', 'reason': set([('path', 'mnt')])},
            {'call': 'acct', 'reason': set([('path', 'mnt')])},
            {'call': 'open', 'reason': set([('path', 'mnt')])},
            {'call': 'dup', 'reason': set([('path', 'mnt')])},
            {'call': 'setns', 'reason': set([('path', 'mnt')])},
            {'call': 'shmctl', 'reason': set([('path', 'mnt')])},
            {'call': 'swapon', 'reason': set([('path', 'mnt')])},
            {'call': 'mmap_pgoff', 'reason': set([('path', 'mnt')])},
            {'call': 'mq_open', 'reason': set([('path', 'mnt')])},
            {'call': 'open_by_handle_at', 'reason': set([('path', 'mnt')])}],
 'listxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
               {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
               {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
               {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
               {'call': 'remap_file_pages',
                'reason': set([('path', 'dentry')])},
               {'call': 'dup3', 'reason': set([('path', 'dentry')])},
               {'call': 'unshare', 'reason': set([('path', 'dentry')])},
               {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
               {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
               {'call': 'flock', 'reason': set([('path', 'dentry')])},
               {'call': 'openat', 'reason': set([('path', 'dentry')])},
               {'call': 'lookup_dcookie',
                'reason': set([('path', 'dentry')])},
               {'call': 'uselib', 'reason': set([('path', 'dentry')])},
               {'call': 'accept4', 'reason': set([('path', 'dentry')])},
               {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
               {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
               {'call': 'shmat', 'reason': set([('path', 'dentry')])},
               {'call': 'socket', 'reason': set([('path', 'dentry')])},
               {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
               {'call': 'perf_event_open',
                'reason': set([('path', 'dentry')])},
               {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
               {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
               {'call': 'acct', 'reason': set([('path', 'dentry')])},
               {'call': 'open', 'reason': set([('path', 'dentry')])},
               {'call': 'dup', 'reason': set([('path', 'dentry')])},
               {'call': 'setns', 'reason': set([('path', 'dentry')])},
               {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
               {'call': 'swapon', 'reason': set([('path', 'dentry')])},
               {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
               {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
               {'call': 'open_by_handle_at',
                'reason': set([('path', 'dentry')])}],
 'llistxattr': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
                {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
                {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
                {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
                {'call': 'remap_file_pages',
                 'reason': set([('path', 'dentry')])},
                {'call': 'dup3', 'reason': set([('path', 'dentry')])},
```



```
                {'call': 'unshare', 'reason': set([('path', 'dentry')])},
                {'call': 'epoll_create1',
                 'reason': set([('path', 'dentry')])},
                {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
                {'call': 'flock', 'reason': set([('path', 'dentry')])},
                {'call': 'openat', 'reason': set([('path', 'dentry')])},
                {'call': 'lookup_dcookie',
                 'reason': set([('path', 'dentry')])},
                {'call': 'uselib', 'reason': set([('path', 'dentry')])},
                {'call': 'accept4', 'reason': set([('path', 'dentry')])},
                {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
                {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
                {'call': 'shmat', 'reason': set([('path', 'dentry')])},
                {'call': 'socket', 'reason': set([('path', 'dentry')])},
                {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
                {'call': 'perf_event_open',
                 'reason': set([('path', 'dentry')])},
                {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
                {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
                {'call': 'acct', 'reason': set([('path', 'dentry')])},
                {'call': 'open', 'reason': set([('path', 'dentry')])},
                {'call': 'dup', 'reason': set([('path', 'dentry')])},
                {'call': 'setns', 'reason': set([('path', 'dentry')])},
                {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
                {'call': 'swapon', 'reason': set([('path', 'dentry')])},
                {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
                {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('path', 'dentry')])}],
 'llseek': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
            {'call': 'flock', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'lremovexattr': [{'call': 'eventfd2',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'swapoff',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'pivot_root',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'memfd_create',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'remap_file_pages',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'dup3',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'unshare',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'epoll_create1',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'epoll_ctl',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'flock',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'openat',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'lookup_dcookie',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'uselib',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'accept4',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'socketpair',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'getcwd',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'shmat',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```



```
                  {'call': 'socket',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'pipe2',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'perf_event_open',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'shmdt',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'quotactl',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'acct',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'open',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'dup',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'setns',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'shmctl',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'swapon',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'mmap_pgoff',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'mq_open',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                  {'call': 'open_by_handle_at',
                   'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'lseek': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
           {'call': 'readahead', 'reason': set([('fd', 'flags')])},
           {'call': 'getdents', 'reason': set([('fd', 'flags')])},
           {'call': 'writev', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
           {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
           {'call': 'pread64', 'reason': set([('fd', 'flags')])},
           {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
           {'call': 'read', 'reason': set([('fd', 'flags')])},
           {'call': 'fchown', 'reason': set([('fd', 'flags')])},
           {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
           {'call': 'utime', 'reason': set([('fd', 'flags')])},
           {'call': 'fsync', 'reason': set([('fd', 'flags')])},
           {'call': 'bpf', 'reason': set([('fd', 'flags')])},
           {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
           {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
           {'call': 'sendto', 'reason': set([('fd', 'flags')])},
           {'call': 'tee', 'reason': set([('fd', 'flags')])},
           {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
           {'call': 'connect', 'reason': set([('fd', 'flags')])},
           {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
           {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
           {'call': 'flock', 'reason': set([('fd', 'flags')])},
           {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
           {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
           {'call': 'accept4', 'reason': set([('fd', 'flags')])},
           {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
           {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
           {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
           {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
           {'call': 'splice', 'reason': set([('fd', 'flags')])},
           {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv', 'reason': set([('fd', 'flags')])},
           {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
```

```
6462            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
6463            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
6464            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
6465            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
6466            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
6467            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
6468            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
6469            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
6470            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
6471            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
6472            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
6473            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
6474            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
6475            {'call': 'listen', 'reason': set([('fd', 'flags')])},
6476            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
6477            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
6478            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
6479            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
6480            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
6481            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
6482            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
6483            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
6484            {'call': 'readv', 'reason': set([('fd', 'flags')])},
6485            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
6486            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
6487            {'call': 'write', 'reason': set([('fd', 'flags')])},
6488            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
6489            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
6490            {'call': 'bind', 'reason': set([('fd', 'flags')])},
6491            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
6492            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
6493  'lsetxattr': [{'call': 'eventfd2',
6494                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6495                {'call': 'swapoff',
6496                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6497                {'call': 'pivot_root',
6498                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6499                {'call': 'memfd_create',
6500                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6501                {'call': 'remap_file_pages',
6502                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6503                {'call': 'dup3',
6504                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6505                {'call': 'unshare',
6506                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6507                {'call': 'epoll_create1',
6508                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6509                {'call': 'epoll_ctl',
6510                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6511                {'call': 'flock',
6512                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6513                {'call': 'openat',
6514                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6515                {'call': 'lookup_dcookie',
6516                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6517                {'call': 'uselib',
6518                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6519                {'call': 'accept4',
6520                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6521                {'call': 'socketpair',
6522                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6523                {'call': 'getcwd',
6524                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6525                {'call': 'shmat',
6526                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6527                {'call': 'socket',
```



```
6528                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6529                {'call': 'pipe2',
6530                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6531                {'call': 'perf_event_open',
6532                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6533                {'call': 'shmdt',
6534                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6535                {'call': 'quotactl',
6536                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6537                {'call': 'acct',
6538                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6539                {'call': 'open',
6540                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6541                {'call': 'dup',
6542                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6543                {'call': 'setns',
6544                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6545                {'call': 'shmctl',
6546                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6547                {'call': 'swapon',
6548                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6549                {'call': 'mmap_pgoff',
6550                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6551                {'call': 'mq_open',
6552                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
6553                {'call': 'open_by_handle_at',
6554                 'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
6555  'lstat': [{'call': 'stat',
6556             'reason': set([('__old_kernel_stat', 'st_ino'),
6557                            ('__old_kernel_stat', 'st_nlink')])},
6558            {'call': 'fstat',
6559             'reason': set([('__old_kernel_stat', 'st_ino'),
6560                            ('__old_kernel_stat', 'st_nlink'),
6561                            ('kstat', 'dev'),
6562                            ('kstat', 'ino'),
6563                            ('kstat', 'nlink'),
6564                            ('kstat', 'rdev')])},
6565            {'call': 'newfstat',
6566             'reason': set([('kstat', 'dev'),
6567                            ('kstat', 'ino'),
6568                            ('kstat', 'nlink'),
6569                            ('kstat', 'rdev')])}],
6570  'madvise': [{'call': 'remap_file_pages',
6571               'reason': set([('vm_area_struct', 'vm_end'),
6572                              ('vm_area_struct', 'vm_start')])},
6573              {'call': 'shmdt',
6574               'reason': set([('vm_area_struct', 'vm_end'),
6575                              ('vm_area_struct', 'vm_start')])},
6576              {'call': 'brk',
6577               'reason': set([('vm_area_struct', 'vm_end'),
6578                              ('vm_area_struct', 'vm_start')])},
6579              {'call': 'get_mempolicy',
6580               'reason': set([('vm_area_struct', 'vm_end'),
6581                              ('vm_area_struct', 'vm_start')])},
6582              {'call': 'munlockall',
6583               'reason': set([('vm_area_struct', 'vm_end'),
6584                              ('vm_area_struct', 'vm_start')])},
6585              {'call': 'pkey_mprotect',
6586               'reason': set([('vm_area_struct', 'vm_end'),
6587                              ('vm_area_struct', 'vm_start')])},
6588              {'call': 'mprotect',
6589               'reason': set([('vm_area_struct', 'vm_end'),
6590                              ('vm_area_struct', 'vm_start')])},
6591              {'call': 'mremap',
6592               'reason': set([('vm_area_struct', 'vm_end'),
6593                              ('vm_area_struct', 'vm_start')])},
```

```
6594              {'call': 'prctl',
6595               'reason': set([('vm_area_struct', 'vm_end'),
6596                              ('vm_area_struct', 'vm_start')])},
6597              {'call': 'munlock',
6598               'reason': set([('vm_area_struct', 'vm_end'),
6599                              ('vm_area_struct', 'vm_start')])},
6600              {'call': 'mincore',
6601               'reason': set([('vm_area_struct', 'vm_end'),
6602                              ('vm_area_struct', 'vm_start')])},
6603              {'call': 'mlockall',
6604               'reason': set([('vm_area_struct', 'vm_end'),
6605                              ('vm_area_struct', 'vm_start')])}],
6606   'migrate_pages': [{'call': 'keyctl',
6607                      'reason': set([('mm_segment_t', 'seg'),
6608                                     ('task_struct', 'cred'),
6609                                     ('task_struct', 'real_cred')])},
6610                     {'call': 'rt_sigtimedwait',
6611                      'reason': set([('mm_segment_t', 'seg'),
6612                                     ('task_struct', 'cred'),
6613                                     ('task_struct', 'real_cred')])},
6614                     {'call': 'msgrcv',
6615                      'reason': set([('mm_segment_t', 'seg'),
6616                                     ('task_struct', 'cred'),
6617                                     ('task_struct', 'real_cred')])},
6618                     {'call': 'kill',
6619                      'reason': set([('mm_segment_t', 'seg'),
6620                                     ('task_struct', 'cred'),
6621                                     ('task_struct', 'real_cred')])},
6622                     {'call': 'sched_getaffinity',
6623                      'reason': set([('mm_segment_t', 'seg'),
6624                                     ('task_struct', 'cred'),
6625                                     ('task_struct', 'real_cred')])},
6626                     {'call': 'sched_setparam',
6627                      'reason': set([('mm_segment_t', 'seg'),
6628                                     ('task_struct', 'cred'),
6629                                     ('task_struct', 'real_cred')])},
6630                     {'call': 'ioprio_set',
6631                      'reason': set([('mm_segment_t', 'seg'),
6632                                     ('task_struct', 'cred'),
6633                                     ('task_struct', 'real_cred')])},
6634                     {'call': 'getppid',
6635                      'reason': set([('mm_segment_t', 'seg'),
6636                                     ('task_struct', 'cred'),
6637                                     ('task_struct', 'real_cred')])},
6638                     {'call': 'ioperm',
6639                      'reason': set([('mm_segment_t', 'seg')])},
6640                     {'call': 'mq_timedreceive',
6641                      'reason': set([('mm_segment_t', 'seg'),
6642                                     ('task_struct', 'cred'),
6643                                     ('task_struct', 'real_cred')])},
6644                     {'call': 'capget',
6645                      'reason': set([('mm_segment_t', 'seg'),
6646                                     ('task_struct', 'cred'),
6647                                     ('task_struct', 'real_cred')])},
6648                     {'call': 'sched_setaffinity',
6649                      'reason': set([('mm_segment_t', 'seg'),
6650                                     ('task_struct', 'cred'),
6651                                     ('task_struct', 'real_cred')])},
6652                     {'call': 'signal',
6653                      'reason': set([('mm_segment_t', 'seg'),
6654                                     ('task_struct', 'cred'),
6655                                     ('task_struct', 'real_cred')])},
6656                     {'call': 'setreuid',
6657                      'reason': set([('cred', 'euid'),
6658                                     ('cred', 'suid'),
6659                                     ('cred', 'uid')])},
```



```
6660                     {'call': 'semtimedop',
6661                      'reason': set([('mm_segment_t', 'seg'),
6662                                     ('task_struct', 'cred'),
6663                                     ('task_struct', 'real_cred')])},
6664                     {'call': 'umount',
6665                      'reason': set([('mm_segment_t', 'seg'),
6666                                     ('task_struct', 'cred'),
6667                                     ('task_struct', 'real_cred')])},
6668                     {'call': 'sched_rr_get_interval',
6669                      'reason': set([('mm_segment_t', 'seg'),
6670                                     ('task_struct', 'cred'),
6671                                     ('task_struct', 'real_cred')])},
6672                     {'call': 'rt_sigprocmask',
6673                      'reason': set([('mm_segment_t', 'seg'),
6674                                     ('task_struct', 'cred'),
6675                                     ('task_struct', 'real_cred')])},
6676                     {'call': 'setsid',
6677                      'reason': set([('mm_segment_t', 'seg'),
6678                                     ('task_struct', 'cred'),
6679                                     ('task_struct', 'real_cred')])},
6680                     {'call': 'sigaltstack',
6681                      'reason': set([('mm_segment_t', 'seg'),
6682                                     ('task_struct', 'cred'),
6683                                     ('task_struct', 'real_cred')])},
6684                     {'call': 'sched_setattr',
6685                      'reason': set([('mm_segment_t', 'seg'),
6686                                     ('task_struct', 'cred'),
6687                                     ('task_struct', 'real_cred')])},
6688                     {'call': 'getitimer',
6689                      'reason': set([('mm_segment_t', 'seg'),
6690                                     ('task_struct', 'cred'),
6691                                     ('task_struct', 'real_cred')])},
6692                     {'call': 'setpgid',
6693                      'reason': set([('mm_segment_t', 'seg'),
6694                                     ('task_struct', 'cred'),
6695                                     ('task_struct', 'real_cred')])},
6696                     {'call': 'getsid',
6697                      'reason': set([('mm_segment_t', 'seg'),
6698                                     ('task_struct', 'cred'),
6699                                     ('task_struct', 'real_cred')])},
6700                     {'call': 'prlimit64',
6701                      'reason': set([('mm_segment_t', 'seg'),
6702                                     ('task_struct', 'cred'),
6703                                     ('task_struct', 'real_cred')])},
6704                     {'call': 'perf_event_open',
6705                      'reason': set([('mm_segment_t', 'seg'),
6706                                     ('task_struct', 'cred'),
6707                                     ('task_struct', 'real_cred')])},
6708                     {'call': 'rt_sigaction',
6709                      'reason': set([('mm_segment_t', 'seg'),
6710                                     ('task_struct', 'cred'),
6711                                     ('task_struct', 'real_cred')])},
6712                     {'call': 'getpgid',
6713                      'reason': set([('mm_segment_t', 'seg'),
6714                                     ('task_struct', 'cred'),
6715                                     ('task_struct', 'real_cred')])},
6716                     {'call': 'getpriority',
6717                      'reason': set([('mm_segment_t', 'seg'),
6718                                     ('task_struct', 'cred'),
6719                                     ('task_struct', 'real_cred')])},
6720                     {'call': 'sigaction',
6721                      'reason': set([('mm_segment_t', 'seg'),
6722                                     ('task_struct', 'cred'),
6723                                     ('task_struct', 'real_cred')])},
6724                     {'call': 'setns',
6725                      'reason': set([('mm_segment_t', 'seg'),
```

```
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'fork',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'get_robust_list',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'ptrace',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'sched_getattr',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'getrusage',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'setresuid',
                     'reason': set([('cred', 'euid'),
                                    ('cred', 'suid'),
                                    ('cred', 'uid')])},
                    {'call': 'setitimer',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'ioprio_get',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'vfork',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'setuid',
                     'reason': set([('cred', 'euid'),
                                    ('cred', 'suid'),
                                    ('cred', 'uid')])},
                    {'call': 'prctl',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'move_pages',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'setpriority',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
```



```
                    {'call': 'clone',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])},
                    {'call': 'sched_getparam',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'cred'),
                                    ('task_struct', 'real_cred')])}],
 'mincore': [{'call': 'keyctl',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'rt_sigtimedwait',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'msgrcv',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'kill',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_getaffinity',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_setparam',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'ioprio_set',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'remap_file_pages',
              'reason': set([('vm_area_struct', 'vm_start')])},
             {'call': 'getppid',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'mq_timedreceive',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'capget',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_setaffinity',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'signal',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'semtimedop',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'umount',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_rr_get_interval',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'rt_sigprocmask',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'setsid',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sigaltstack',
              'reason': set([('mm_segment_t', 'seg'),
                             ('task_struct', 'mm')])},
             {'call': 'sched_setattr',
```

```
6858                'reason': set([('mm_segment_t', 'seg'),
6859                               ('task_struct', 'mm')])},
6860               {'call': 'migrate_pages',
6861                'reason': set([('mm_segment_t', 'seg'),
6862                               ('task_struct', 'mm')])},
6863               {'call': 'getitimer',
6864                'reason': set([('mm_segment_t', 'seg'),
6865                               ('task_struct', 'mm')])},
6866               {'call': 'setpgid',
6867                'reason': set([('mm_segment_t', 'seg'),
6868                               ('task_struct', 'mm')])},
6869               {'call': 'getsid',
6870                'reason': set([('mm_segment_t', 'seg'),
6871                               ('task_struct', 'mm')])},
6872               {'call': 'prlimit64',
6873                'reason': set([('mm_segment_t', 'seg'),
6874                               ('task_struct', 'mm')])},
6875               {'call': 'perf_event_open',
6876                'reason': set([('mm_segment_t', 'seg'),
6877                               ('task_struct', 'mm')])},
6878               {'call': 'shmdt',
6879                'reason': set([('vm_area_struct', 'vm_start')])},
6880               {'call': 'rt_sigaction',
6881                'reason': set([('mm_segment_t', 'seg'),
6882                               ('task_struct', 'mm')])},
6883               {'call': 'getpgid',
6884                'reason': set([('mm_segment_t', 'seg'),
6885                               ('task_struct', 'mm')])},
6886               {'call': 'brk', 'reason': set([('vm_area_struct', 'vm_start')])},
6887               {'call': 'getpriority',
6888                'reason': set([('mm_segment_t', 'seg'),
6889                               ('task_struct', 'mm')])},
6890               {'call': 'sigaction',
6891                'reason': set([('mm_segment_t', 'seg'),
6892                               ('task_struct', 'mm')])},
6893               {'call': 'setns',
6894                'reason': set([('mm_segment_t', 'seg'),
6895                               ('task_struct', 'mm')])},
6896               {'call': 'fork',
6897                'reason': set([('mm_segment_t', 'seg'),
6898                               ('task_struct', 'mm')])},
6899               {'call': 'get_mempolicy',
6900                'reason': set([('vm_area_struct', 'vm_start')])},
6901               {'call': 'get_robust_list',
6902                'reason': set([('mm_segment_t', 'seg'),
6903                               ('task_struct', 'mm')])},
6904               {'call': 'mq_timedsend',
6905                'reason': set([('mm_segment_t', 'seg'),
6906                               ('task_struct', 'mm')])},
6907               {'call': 'sched_getscheduler',
6908                'reason': set([('mm_segment_t', 'seg'),
6909                               ('task_struct', 'mm')])},
6910               {'call': 'ptrace',
6911                'reason': set([('mm_segment_t', 'seg'),
6912                               ('task_struct', 'mm')])},
6913               {'call': 'munlockall',
6914                'reason': set([('vm_area_struct', 'vm_start')])},
6915               {'call': 'pkey_mprotect',
6916                'reason': set([('vm_area_struct', 'vm_start')])},
6917               {'call': 'madvise',
6918                'reason': set([('vm_area_struct', 'vm_start')])},
6919               {'call': 'sched_getattr',
6920                'reason': set([('mm_segment_t', 'seg'),
6921                               ('task_struct', 'mm')])},
6922               {'call': 'getrusage',
6923                'reason': set([('mm_segment_t', 'seg'),
```



```
6924                               ('task_struct', 'mm')])},
6925               {'call': 'sched_setscheduler',
6926                'reason': set([('mm_segment_t', 'seg'),
6927                               ('task_struct', 'mm')])},
6928               {'call': 'setitimer',
6929                'reason': set([('mm_segment_t', 'seg'),
6930                               ('task_struct', 'mm')])},
6931               {'call': 'ioprio_get',
6932                'reason': set([('mm_segment_t', 'seg'),
6933                               ('task_struct', 'mm')])},
6934               {'call': 'vfork',
6935                'reason': set([('mm_segment_t', 'seg'),
6936                               ('task_struct', 'mm')])},
6937               {'call': 'mprotect',
6938                'reason': set([('vm_area_struct', 'vm_start')])},
6939               {'call': 'mremap',
6940                'reason': set([('vm_area_struct', 'vm_start')])},
6941               {'call': 'prctl',
6942                'reason': set([('mm_segment_t', 'seg'),
6943                               ('task_struct', 'mm'),
6944                               ('vm_area_struct', 'vm_start')])},
6945               {'call': 'move_pages',
6946                'reason': set([('mm_segment_t', 'seg'),
6947                               ('task_struct', 'mm')])},
6948               {'call': 'munlock',
6949                'reason': set([('vm_area_struct', 'vm_start')])},
6950               {'call': 'setpriority',
6951                'reason': set([('mm_segment_t', 'seg'),
6952                               ('task_struct', 'mm')])},
6953               {'call': 'clone',
6954                'reason': set([('mm_segment_t', 'seg'),
6955                               ('task_struct', 'mm')])},
6956               {'call': 'sched_getparam',
6957                'reason': set([('mm_segment_t', 'seg'),
6958                               ('task_struct', 'mm')])},
6959               {'call': 'mlockall',
6960                'reason': set([('vm_area_struct', 'vm_start')])}],
6961  'mkdirat': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
6962              {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
6963              {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
6964              {'call': 'quotactl',
6965               'reason': set([('super_block', 's_flags')])},
6966              {'call': 'swapon', 'reason': set([('super_block', 's_flags')])}],
6967  'mknodat': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
6968              {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
6969              {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
6970              {'call': 'quotactl',
6971               'reason': set([('super_block', 's_flags')])},
6972              {'call': 'swapon', 'reason': set([('super_block', 's_flags')])}],
6973  'mlock': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
6974            {'call': 'rt_sigtimedwait',
6975             'reason': set([('task_struct', 'mm')])},
6976            {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
6977            {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
6978            {'call': 'sched_getaffinity',
6979             'reason': set([('task_struct', 'mm')])},
6980            {'call': 'sched_setparam', 'reason': set([('task_struct', 'mm')])},
6981            {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
6982            {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
6983            {'call': 'mq_timedreceive',
6984             'reason': set([('task_struct', 'mm')])},
6985            {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
6986            {'call': 'sched_setaffinity',
6987             'reason': set([('task_struct', 'mm')])},
6988            {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
6989            {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
```

```
6990            {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
6991            {'call': 'sched_rr_get_interval',
6992             'reason': set([('task_struct', 'mm')])},
6993            {'call': 'rt_sigprocmask', 'reason': set([('task_struct', 'mm')])},
6994            {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
6995            {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
6996            {'call': 'sched_setattr', 'reason': set([('task_struct', 'mm')])},
6997            {'call': 'migrate_pages', 'reason': set([('task_struct', 'mm')])},
6998            {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
6999            {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
7000            {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
7001            {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
7002            {'call': 'perf_event_open',
7003             'reason': set([('task_struct', 'mm')])},
7004            {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
7005            {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
7006            {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
7007            {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
7008            {'call': 'setns', 'reason': set([('task_struct', 'mm')])},
7009            {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
7010            {'call': 'get_robust_list',
7011             'reason': set([('task_struct', 'mm')])},
7012            {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
7013            {'call': 'sched_getscheduler',
7014             'reason': set([('task_struct', 'mm')])},
7015            {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
7016            {'call': 'sched_getattr', 'reason': set([('task_struct', 'mm')])},
7017            {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
7018            {'call': 'sched_setscheduler',
7019             'reason': set([('task_struct', 'mm')])},
7020            {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
7021            {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
7022            {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
7023            {'call': 'prctl', 'reason': set([('task_struct', 'mm')])},
7024            {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
7025            {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
7026            {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
7027            {'call': 'sched_getparam', 'reason': set([('task_struct', 'mm')])}],
7028  'mlock2': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
7029             {'call': 'rt_sigtimedwait',
7030              'reason': set([('task_struct', 'mm')])},
7031             {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
7032             {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
7033             {'call': 'sched_getaffinity',
7034              'reason': set([('task_struct', 'mm')])},
7035             {'call': 'sched_setparam',
7036              'reason': set([('task_struct', 'mm')])},
7037             {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
7038             {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
7039             {'call': 'mq_timedreceive',
7040              'reason': set([('task_struct', 'mm')])},
7041             {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
7042             {'call': 'sched_setaffinity',
7043              'reason': set([('task_struct', 'mm')])},
7044             {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
7045             {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
7046             {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
7047             {'call': 'sched_rr_get_interval',
7048              'reason': set([('task_struct', 'mm')])},
7049             {'call': 'rt_sigprocmask',
7050              'reason': set([('task_struct', 'mm')])},
7051             {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
7052             {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
7053             {'call': 'sched_setattr', 'reason': set([('task_struct', 'mm')])},
7054             {'call': 'migrate_pages', 'reason': set([('task_struct', 'mm')])},
7055             {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
```



```
7056            {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
7057            {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
7058            {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
7059            {'call': 'perf_event_open',
7060             'reason': set([('task_struct', 'mm')])},
7061            {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
7062            {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
7063            {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
7064            {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
7065            {'call': 'setns', 'reason': set([('task_struct', 'mm')])},
7066            {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
7067            {'call': 'get_robust_list',
7068             'reason': set([('task_struct', 'mm')])},
7069            {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
7070            {'call': 'sched_getscheduler',
7071             'reason': set([('task_struct', 'mm')])},
7072            {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
7073            {'call': 'sched_getattr', 'reason': set([('task_struct', 'mm')])},
7074            {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
7075            {'call': 'sched_setscheduler',
7076             'reason': set([('task_struct', 'mm')])},
7077            {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
7078            {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
7079            {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
7080            {'call': 'prctl', 'reason': set([('task_struct', 'mm')])},
7081            {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
7082            {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
7083            {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
7084            {'call': 'sched_getparam',
7085             'reason': set([('task_struct', 'mm')])}],
7086  'mlockall': [{'call': 'keyctl',
7087                'reason': set([('task_struct', 'personality')])},
7088               {'call': 'rt_sigtimedwait',
7089                'reason': set([('task_struct', 'personality')])},
7090               {'call': 'msgrcv',
7091                'reason': set([('task_struct', 'personality')])},
7092               {'call': 'kill',
7093                'reason': set([('task_struct', 'personality')])},
7094               {'call': 'swapoff', 'reason': set([('mm_struct', 'total_vm')])},
7095               {'call': 'sched_getaffinity',
7096                'reason': set([('task_struct', 'personality')])},
7097               {'call': 'sched_setparam',
7098                'reason': set([('task_struct', 'personality')])},
7099               {'call': 'ioprio_set',
7100                'reason': set([('task_struct', 'personality')])},
7101               {'call': 'personality',
7102                'reason': set([('task_struct', 'personality')])},
7103               {'call': 'remap_file_pages',
7104                'reason': set([('mm_struct', 'total_vm'),
7105                               ('vm_area_struct', 'vm_end'),
7106                               ('vm_area_struct', 'vm_start')])},
7107               {'call': 'io_getevents',
7108                'reason': set([('mm_struct', 'total_vm')])},
7109               {'call': 'getppid',
7110                'reason': set([('task_struct', 'personality')])},
7111               {'call': 'mq_timedreceive',
7112                'reason': set([('task_struct', 'personality')])},
7113               {'call': 'capget',
7114                'reason': set([('task_struct', 'personality')])},
7115               {'call': 'sched_setaffinity',
7116                'reason': set([('task_struct', 'personality')])},
7117               {'call': 'signal',
7118                'reason': set([('task_struct', 'personality')])},
7119               {'call': 'semtimedop',
7120                'reason': set([('task_struct', 'personality')])},
7121               {'call': 'umount',
```

```
7122                'reason': set([('task_struct', 'personality')])},
7123               {'call': 'sched_rr_get_interval',
7124                'reason': set([('task_struct', 'personality')])},
7125               {'call': 'rt_sigprocmask',
7126                'reason': set([('task_struct', 'personality')])},
7127               {'call': 'setsid',
7128                'reason': set([('task_struct', 'personality')])},
7129               {'call': 'sigaltstack',
7130                'reason': set([('task_struct', 'personality')])},
7131               {'call': 'sched_setattr',
7132                'reason': set([('task_struct', 'personality')])},
7133               {'call': 'migrate_pages',
7134                'reason': set([('mm_struct', 'total_vm'),
7135                               ('task_struct', 'personality')])},
7136               {'call': 'getitimer',
7137                'reason': set([('task_struct', 'personality')])},
7138               {'call': 'setpgid',
7139                'reason': set([('task_struct', 'personality')])},
7140               {'call': 'getsid',
7141                'reason': set([('task_struct', 'personality')])},
7142               {'call': 'prlimit64',
7143                'reason': set([('task_struct', 'personality')])},
7144               {'call': 'perf_event_open',
7145                'reason': set([('task_struct', 'personality')])},
7146               {'call': 'shmdt',
7147                'reason': set([('mm_struct', 'total_vm'),
7148                               ('vm_area_struct', 'vm_end'),
7149                               ('vm_area_struct', 'vm_start')])},
7150               {'call': 'rt_sigaction',
7151                'reason': set([('task_struct', 'personality')])},
7152               {'call': 'getpgid',
7153                'reason': set([('task_struct', 'personality')])},
7154               {'call': 'brk',
7155                'reason': set([('mm_struct', 'total_vm'),
7156                               ('vm_area_struct', 'vm_end'),
7157                               ('vm_area_struct', 'vm_start')])},
7158               {'call': 'getpriority',
7159                'reason': set([('task_struct', 'personality')])},
7160               {'call': 'sigaction',
7161                'reason': set([('task_struct', 'personality')])},
7162               {'call': 'setns',
7163                'reason': set([('task_struct', 'personality')])},
7164               {'call': 'fork',
7165                'reason': set([('task_struct', 'personality')])},
7166               {'call': 'get_mempolicy',
7167                'reason': set([('mm_struct', 'total_vm'),
7168                               ('vm_area_struct', 'vm_end'),
7169                               ('vm_area_struct', 'vm_start')])},
7170               {'call': 'get_robust_list',
7171                'reason': set([('task_struct', 'personality')])},
7172               {'call': 'mq_timedsend',
7173                'reason': set([('task_struct', 'personality')])},
7174               {'call': 'sched_getscheduler',
7175                'reason': set([('task_struct', 'personality')])},
7176               {'call': 'ptrace',
7177                'reason': set([('task_struct', 'personality')])},
7178               {'call': 'munlockall',
7179                'reason': set([('vm_area_struct', 'vm_end'),
7180                               ('vm_area_struct', 'vm_start')])},
7181               {'call': 'pkey_mprotect',
7182                'reason': set([('vm_area_struct', 'vm_end'),
7183                               ('vm_area_struct', 'vm_start')])},
7184               {'call': 'madvise',
7185                'reason': set([('vm_area_struct', 'vm_end'),
7186                               ('vm_area_struct', 'vm_start')])},
7187               {'call': 'sched_getattr',
```



```
7188                'reason': set([('task_struct', 'personality')])},
7189               {'call': 'getrusage',
7190                'reason': set([('mm_struct', 'total_vm'),
7191                               ('task_struct', 'personality')])},
7192               {'call': 'sched_setscheduler',
7193                'reason': set([('task_struct', 'personality')])},
7194               {'call': 'setitimer',
7195                'reason': set([('task_struct', 'personality')])},
7196               {'call': 'ioprio_get',
7197                'reason': set([('task_struct', 'personality')])},
7198               {'call': 'vfork',
7199                'reason': set([('task_struct', 'personality')])},
7200               {'call': 'io_setup',
7201                'reason': set([('mm_struct', 'total_vm')])},
7202               {'call': 'mprotect',
7203                'reason': set([('vm_area_struct', 'vm_end'),
7204                               ('vm_area_struct', 'vm_start')])},
7205               {'call': 'mremap',
7206                'reason': set([('mm_struct', 'total_vm'),
7207                               ('vm_area_struct', 'vm_end'),
7208                               ('vm_area_struct', 'vm_start')])},
7209               {'call': 'io_destroy',
7210                'reason': set([('mm_struct', 'total_vm')])},
7211               {'call': 'mbind', 'reason': set([('mm_struct', 'total_vm')])},
7212               {'call': 'prctl',
7213                'reason': set([('mm_struct', 'total_vm'),
7214                               ('task_struct', 'personality'),
7215                               ('vm_area_struct', 'vm_end'),
7216                               ('vm_area_struct', 'vm_start')])},
7217               {'call': 'move_pages',
7218                'reason': set([('mm_struct', 'total_vm'),
7219                               ('task_struct', 'personality')])},
7220               {'call': 'modify_ldt',
7221                'reason': set([('mm_struct', 'total_vm')])},
7222               {'call': 'munlock',
7223                'reason': set([('vm_area_struct', 'vm_end'),
7224                               ('vm_area_struct', 'vm_start')])},
7225               {'call': 'setpriority',
7226                'reason': set([('task_struct', 'personality')])},
7227               {'call': 'mincore',
7228                'reason': set([('mm_struct', 'total_vm'),
7229                               ('vm_area_struct', 'vm_end'),
7230                               ('vm_area_struct', 'vm_start')])},
7231               {'call': 'clone',
7232                'reason': set([('task_struct', 'personality')])},
7233               {'call': 'sched_getparam',
7234                'reason': set([('task_struct', 'personality')])},
7235               {'call': 'io_cancel',
7236                'reason': set([('mm_struct', 'total_vm')])}],
7237  'mmap_pgoff': [{'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
7238                 {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
7239                 {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
7240                 {'call': 'remap_file_pages',
7241                  'reason': set([('file', 'f_op')])},
7242                 {'call': 'dup3', 'reason': set([('file', 'f_op')])},
7243                 {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
7244                 {'call': 'epoll_ctl', 'reason': set([('file', 'f_op')])},
7245                 {'call': 'flock', 'reason': set([('file', 'f_op')])},
7246                 {'call': 'openat', 'reason': set([('file', 'f_op')])},
7247                 {'call': 'uselib', 'reason': set([('file', 'f_op')])},
7248                 {'call': 'accept4', 'reason': set([('file', 'f_op')])},
7249                 {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
7250                 {'call': 'shmat', 'reason': set([('file', 'f_op')])},
7251                 {'call': 'socket', 'reason': set([('file', 'f_op')])},
7252                 {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
7253                 {'call': 'perf_event_open',
```

```
                 'reason': set([('file', 'f_op')])},
                {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
                {'call': 'acct', 'reason': set([('file', 'f_op')])},
                {'call': 'open', 'reason': set([('file', 'f_op')])},
                {'call': 'dup', 'reason': set([('file', 'f_op')])},
                {'call': 'setns', 'reason': set([('file', 'f_op')])},
                {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
                {'call': 'swapon', 'reason': set([('file', 'f_op')])},
                {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_op')])}],
 'modify_ldt': [{'call': 'swapoff', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'remap_file_pages',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'io_getevents',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'get_thread_area',
                 'reason': set([('user_desc', 'base_addr'),
                                ('user_desc', 'contents'),
                                ('user_desc', 'entry_number'),
                                ('user_desc', 'limit'),
                                ('user_desc', 'limit_in_pages'),
                                ('user_desc', 'read_exec_only'),
                                ('user_desc', 'seg_32bit'),
                                ('user_desc', 'seg_not_present'),
                                ('user_desc', 'useable')])},
                {'call': 'migrate_pages',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'shmdt', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'brk', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'get_mempolicy',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'getrusage',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'io_setup',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'mremap', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'io_destroy',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'mbind', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'prctl', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'move_pages',
                 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'mincore', 'reason': set([('mm_context_t', 'ldt')])},
                {'call': 'set_thread_area',
                 'reason': set([('user_desc', 'base_addr'),
                                ('user_desc', 'contents'),
                                ('user_desc', 'entry_number'),
                                ('user_desc', 'limit'),
                                ('user_desc', 'limit_in_pages'),
                                ('user_desc', 'read_exec_only'),
                                ('user_desc', 'seg_32bit'),
                                ('user_desc', 'seg_not_present'),
                                ('user_desc', 'useable')])},
                {'call': 'io_cancel',
                 'reason': set([('mm_context_t', 'ldt')])}],
 'mount': [{'call': 'keyctl',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'msgrcv',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_getaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setparam',
```



```
            'reason': set([('task_struct', 'personality')])},
           {'call': 'ioprio_set',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'personality',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getppid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'capget',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'signal',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'semtimedop',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'umount',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_rr_get_interval',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigprocmask',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'setsid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sigaltstack',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setattr',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'migrate_pages',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getitimer',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'setpgid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getsid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'prlimit64',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'perf_event_open',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigaction',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getpgid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getpriority',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sigaction',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
           {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
           {'call': 'get_robust_list',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'mq_timedsend',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_getscheduler',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'ptrace',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_getattr',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'getrusage',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setscheduler',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'setitimer',
```

```
7386              'reason': set([('task_struct', 'personality')])},
7387             {'call': 'ioprio_get',
7388              'reason': set([('task_struct', 'personality')])},
7389             {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
7390             {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
7391             {'call': 'move_pages',
7392              'reason': set([('task_struct', 'personality')])},
7393             {'call': 'setpriority',
7394              'reason': set([('task_struct', 'personality')])},
7395             {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
7396             {'call': 'sched_getparam',
7397              'reason': set([('task_struct', 'personality')])}],
7398  'mprotect': [{'call': 'keyctl',
7399                'reason': set([('task_struct', 'mm'),
7400                               ('task_struct', 'personality')])},
7401               {'call': 'rt_sigtimedwait',
7402                'reason': set([('task_struct', 'mm'),
7403                               ('task_struct', 'personality')])},
7404               {'call': 'msgrcv',
7405                'reason': set([('task_struct', 'mm'),
7406                               ('task_struct', 'personality')])},
7407               {'call': 'kill',
7408                'reason': set([('task_struct', 'mm'),
7409                               ('task_struct', 'personality')])},
7410               {'call': 'sched_getaffinity',
7411                'reason': set([('task_struct', 'mm'),
7412                               ('task_struct', 'personality')])},
7413               {'call': 'sched_setparam',
7414                'reason': set([('task_struct', 'mm'),
7415                               ('task_struct', 'personality')])},
7416               {'call': 'ioprio_set',
7417                'reason': set([('task_struct', 'mm'),
7418                               ('task_struct', 'personality')])},
7419               {'call': 'personality',
7420                'reason': set([('task_struct', 'personality')])},
7421               {'call': 'remap_file_pages',
7422                'reason': set([('vm_area_struct', 'vm_end'),
7423                               ('vm_area_struct', 'vm_flags'),
7424                               ('vm_area_struct', 'vm_start')])},
7425               {'call': 'getppid',
7426                'reason': set([('task_struct', 'mm'),
7427                               ('task_struct', 'personality')])},
7428               {'call': 'mq_timedreceive',
7429                'reason': set([('task_struct', 'mm'),
7430                               ('task_struct', 'personality')])},
7431               {'call': 'capget',
7432                'reason': set([('task_struct', 'mm'),
7433                               ('task_struct', 'personality')])},
7434               {'call': 'sched_setaffinity',
7435                'reason': set([('task_struct', 'mm'),
7436                               ('task_struct', 'personality')])},
7437               {'call': 'signal',
7438                'reason': set([('task_struct', 'mm'),
7439                               ('task_struct', 'personality')])},
7440               {'call': 'semtimedop',
7441                'reason': set([('task_struct', 'mm'),
7442                               ('task_struct', 'personality')])},
7443               {'call': 'umount',
7444                'reason': set([('task_struct', 'mm'),
7445                               ('task_struct', 'personality')])},
7446               {'call': 'sched_rr_get_interval',
7447                'reason': set([('task_struct', 'mm'),
7448                               ('task_struct', 'personality')])},
7449               {'call': 'rt_sigprocmask',
7450                'reason': set([('task_struct', 'mm'),
7451                               ('task_struct', 'personality')])},
```



```
7452               {'call': 'setsid',
7453                'reason': set([('task_struct', 'mm'),
7454                               ('task_struct', 'personality')])},
7455               {'call': 'sigaltstack',
7456                'reason': set([('task_struct', 'mm'),
7457                               ('task_struct', 'personality')])},
7458               {'call': 'sched_setattr',
7459                'reason': set([('task_struct', 'mm'),
7460                               ('task_struct', 'personality')])},
7461               {'call': 'migrate_pages',
7462                'reason': set([('task_struct', 'mm'),
7463                               ('task_struct', 'personality')])},
7464               {'call': 'getitimer',
7465                'reason': set([('task_struct', 'mm'),
7466                               ('task_struct', 'personality')])},
7467               {'call': 'setpgid',
7468                'reason': set([('task_struct', 'mm'),
7469                               ('task_struct', 'personality')])},
7470               {'call': 'getsid',
7471                'reason': set([('task_struct', 'mm'),
7472                               ('task_struct', 'personality')])},
7473               {'call': 'prlimit64',
7474                'reason': set([('task_struct', 'mm'),
7475                               ('task_struct', 'personality')])},
7476               {'call': 'perf_event_open',
7477                'reason': set([('task_struct', 'mm'),
7478                               ('task_struct', 'personality')])},
7479               {'call': 'shmdt',
7480                'reason': set([('vm_area_struct', 'vm_end'),
7481                               ('vm_area_struct', 'vm_flags'),
7482                               ('vm_area_struct', 'vm_start')])},
7483               {'call': 'rt_sigaction',
7484                'reason': set([('task_struct', 'mm'),
7485                               ('task_struct', 'personality')])},
7486               {'call': 'getpgid',
7487                'reason': set([('task_struct', 'mm'),
7488                               ('task_struct', 'personality')])},
7489               {'call': 'brk',
7490                'reason': set([('vm_area_struct', 'vm_end'),
7491                               ('vm_area_struct', 'vm_flags'),
7492                               ('vm_area_struct', 'vm_start')])},
7493               {'call': 'getpriority',
7494                'reason': set([('task_struct', 'mm'),
7495                               ('task_struct', 'personality')])},
7496               {'call': 'sigaction',
7497                'reason': set([('task_struct', 'mm'),
7498                               ('task_struct', 'personality')])},
7499               {'call': 'setns',
7500                'reason': set([('task_struct', 'mm'),
7501                               ('task_struct', 'personality')])},
7502               {'call': 'fork',
7503                'reason': set([('task_struct', 'mm'),
7504                               ('task_struct', 'personality')])},
7505               {'call': 'get_mempolicy',
7506                'reason': set([('vm_area_struct', 'vm_end'),
7507                               ('vm_area_struct', 'vm_flags'),
7508                               ('vm_area_struct', 'vm_start')])},
7509               {'call': 'get_robust_list',
7510                'reason': set([('task_struct', 'mm'),
7511                               ('task_struct', 'personality')])},
7512               {'call': 'mq_timedsend',
7513                'reason': set([('task_struct', 'mm'),
7514                               ('task_struct', 'personality')])},
7515               {'call': 'sched_getscheduler',
7516                'reason': set([('task_struct', 'mm'),
7517                               ('task_struct', 'personality')])},
```

```
7518                {'call': 'ptrace',
7519                 'reason': set([('task_struct', 'mm'),
7520                                ('task_struct', 'personality')])},
7521                {'call': 'munlockall',
7522                 'reason': set([('vm_area_struct', 'vm_end'),
7523                                ('vm_area_struct', 'vm_flags'),
7524                                ('vm_area_struct', 'vm_start')])},
7525                {'call': 'pkey_mprotect',
7526                 'reason': set([('vm_area_struct', 'vm_end'),
7527                                ('vm_area_struct', 'vm_flags'),
7528                                ('vm_area_struct', 'vm_start')])},
7529                {'call': 'madvise',
7530                 'reason': set([('vm_area_struct', 'vm_end'),
7531                                ('vm_area_struct', 'vm_flags'),
7532                                ('vm_area_struct', 'vm_start')])},
7533                {'call': 'sched_getattr',
7534                 'reason': set([('task_struct', 'mm'),
7535                                ('task_struct', 'personality')])},
7536                {'call': 'getrusage',
7537                 'reason': set([('task_struct', 'mm'),
7538                                ('task_struct', 'personality')])},
7539                {'call': 'sched_setscheduler',
7540                 'reason': set([('task_struct', 'mm'),
7541                                ('task_struct', 'personality')])},
7542                {'call': 'setitimer',
7543                 'reason': set([('task_struct', 'mm'),
7544                                ('task_struct', 'personality')])},
7545                {'call': 'ioprio_get',
7546                 'reason': set([('task_struct', 'mm'),
7547                                ('task_struct', 'personality')])},
7548                {'call': 'vfork',
7549                 'reason': set([('task_struct', 'mm'),
7550                                ('task_struct', 'personality')])},
7551                {'call': 'mremap',
7552                 'reason': set([('vm_area_struct', 'vm_end'),
7553                                ('vm_area_struct', 'vm_flags'),
7554                                ('vm_area_struct', 'vm_start')])},
7555                {'call': 'prctl',
7556                 'reason': set([('task_struct', 'mm'),
7557                                ('task_struct', 'personality'),
7558                                ('vm_area_struct', 'vm_end'),
7559                                ('vm_area_struct', 'vm_flags'),
7560                                ('vm_area_struct', 'vm_start')])},
7561                {'call': 'move_pages',
7562                 'reason': set([('task_struct', 'mm'),
7563                                ('task_struct', 'personality')])},
7564                {'call': 'munlock',
7565                 'reason': set([('vm_area_struct', 'vm_end'),
7566                                ('vm_area_struct', 'vm_flags'),
7567                                ('vm_area_struct', 'vm_start')])},
7568                {'call': 'setpriority',
7569                 'reason': set([('task_struct', 'mm'),
7570                                ('task_struct', 'personality')])},
7571                {'call': 'mincore',
7572                 'reason': set([('vm_area_struct', 'vm_end'),
7573                                ('vm_area_struct', 'vm_flags'),
7574                                ('vm_area_struct', 'vm_start')])},
7575                {'call': 'clone',
7576                 'reason': set([('task_struct', 'mm'),
7577                                ('task_struct', 'personality')])},
7578                {'call': 'sched_getparam',
7579                 'reason': set([('task_struct', 'mm'),
7580                                ('task_struct', 'personality')])},
7581                {'call': 'mlockall',
7582                 'reason': set([('vm_area_struct', 'vm_end'),
7583                                ('vm_area_struct', 'vm_flags'),
```



```
7584                                ('vm_area_struct', 'vm_start')])}],
7585  'mq_getsetattr': [{'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
7586                    {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
7587                    {'call': 'memfd_create',
7588                     'reason': set([('file', 'f_op')])},
7589                    {'call': 'remap_file_pages',
7590                     'reason': set([('file', 'f_op')])},
7591                    {'call': 'dup3', 'reason': set([('file', 'f_op')])},
7592                    {'call': 'mq_timedreceive',
7593                     'reason': set([('mq_attr', 'mq_flags')])},
7594                    {'call': 'epoll_create1',
7595                     'reason': set([('file', 'f_op')])},
7596                    {'call': 'epoll_ctl', 'reason': set([('file', 'f_op')])},
7597                    {'call': 'flock', 'reason': set([('file', 'f_op')])},
7598                    {'call': 'openat', 'reason': set([('file', 'f_op')])},
7599                    {'call': 'uselib', 'reason': set([('file', 'f_op')])},
7600                    {'call': 'accept4', 'reason': set([('file', 'f_op')])},
7601                    {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
7602                    {'call': 'shmat', 'reason': set([('file', 'f_op')])},
7603                    {'call': 'socket', 'reason': set([('file', 'f_op')])},
7604                    {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
7605                    {'call': 'perf_event_open',
7606                     'reason': set([('file', 'f_op')])},
7607                    {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
7608                    {'call': 'acct', 'reason': set([('file', 'f_op')])},
7609                    {'call': 'open', 'reason': set([('file', 'f_op')])},
7610                    {'call': 'dup', 'reason': set([('file', 'f_op')])},
7611                    {'call': 'setns', 'reason': set([('file', 'f_op')])},
7612                    {'call': 'mq_timedsend',
7613                     'reason': set([('mq_attr', 'mq_flags')])},
7614                    {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
7615                    {'call': 'swapon', 'reason': set([('file', 'f_op')])},
7616                    {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
7617                    {'call': 'mq_notify',
7618                     'reason': set([('mq_attr', 'mq_flags')])},
7619                    {'call': 'mq_open',
7620                     'reason': set([('file', 'f_op'),
7621                                    ('mq_attr', 'mq_flags')])},
7622                    {'call': 'open_by_handle_at',
7623                     'reason': set([('file', 'f_op')])}],
7624  'mq_notify': [{'call': 'rt_sigtimedwait',
7625                 'reason': set([('sigval', 'sival_ptr')])},
7626                {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
7627                {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
7628                {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
7629                {'call': 'remap_file_pages',
7630                 'reason': set([('file', 'f_op')])},
7631                {'call': 'dup3', 'reason': set([('file', 'f_op')])},
7632                {'call': 'mq_timedreceive',
7633                 'reason': set([('mqueue_inode_info', 'notify_owner'),
7634                                ('sigevent', 'sigev_notify'),
7635                                ('sigevent', 'sigev_signo'),
7636                                ('sigval', 'sival_ptr')])},
7637                {'call': 'timer_create',
7638                 'reason': set([('sigevent', 'sigev_notify'),
7639                                ('sigevent', 'sigev_signo'),
7640                                ('sigval', 'sival_ptr')])},
7641                {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
7642                {'call': 'rt_sigqueueinfo',
7643                 'reason': set([('sigval', 'sival_ptr')])},
7644                {'call': 'epoll_ctl', 'reason': set([('file', 'f_op')])},
7645                {'call': 'flock', 'reason': set([('file', 'f_op')])},
7646                {'call': 'tgkill', 'reason': set([('sigval', 'sival_ptr')])},
7647                {'call': 'openat', 'reason': set([('file', 'f_op')])},
7648                {'call': 'uselib', 'reason': set([('file', 'f_op')])},
7649                {'call': 'accept4', 'reason': set([('file', 'f_op')])},
```

```
7650                {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
7651                {'call': 'shmat', 'reason': set([('file', 'f_op')])},
7652                {'call': 'socket', 'reason': set([('file', 'f_op')])},
7653                {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
7654                {'call': 'perf_event_open', 'reason': set([('file', 'f_op')])},
7655                {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
7656                {'call': 'acct', 'reason': set([('file', 'f_op')])},
7657                {'call': 'open', 'reason': set([('file', 'f_op')])},
7658                {'call': 'rt_tgsigqueueinfo',
7659                 'reason': set([('sigval', 'sival_ptr')])},
7660                {'call': 'mq_getsetattr',
7661                 'reason': set([('mqueue_inode_info', 'notify_owner'),
7662                                ('sigevent', 'sigev_notify'),
7663                                ('sigevent', 'sigev_signo'),
7664                                ('sigval', 'sival_ptr')])},
7665                {'call': 'dup', 'reason': set([('file', 'f_op')])},
7666                {'call': 'setns', 'reason': set([('file', 'f_op')])},
7667                {'call': 'mq_timedsend',
7668                 'reason': set([('mqueue_inode_info', 'notify_owner'),
7669                                ('sigevent', 'sigev_notify'),
7670                                ('sigevent', 'sigev_signo'),
7671                                ('sigval', 'sival_ptr')])},
7672                {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
7673                {'call': 'swapon', 'reason': set([('file', 'f_op')])},
7674                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
7675                {'call': 'rt_sigreturn',
7676                 'reason': set([('sigval', 'sival_ptr')])},
7677                {'call': 'tkill', 'reason': set([('sigval', 'sival_ptr')])},
7678                {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
7679                {'call': 'open_by_handle_at',
7680                 'reason': set([('file', 'f_op')])}],
7681  'mq_open': [{'call': 'sysfs', 'reason': set([('filename', 'name')])},
7682              {'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
7683              {'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
7684              {'call': 'swapoff',
7685               'reason': set([('filename', 'name'), ('path', 'dentry')])},
7686              {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
7687              {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
7688              {'call': 'remap_file_pages',
7689               'reason': set([('path', 'dentry')])},
7690              {'call': 'dup3', 'reason': set([('path', 'dentry')])},
7691              {'call': 'unshare', 'reason': set([('path', 'dentry')])},
7692              {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
7693              {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
7694              {'call': 'flock', 'reason': set([('path', 'dentry')])},
7695              {'call': 'openat',
7696               'reason': set([('filename', 'name'), ('path', 'dentry')])},
7697              {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
7698              {'call': 'uselib',
7699               'reason': set([('filename', 'name'), ('path', 'dentry')])},
7700              {'call': 'renameat2', 'reason': set([('filename', 'name')])},
7701              {'call': 'accept4', 'reason': set([('path', 'dentry')])},
7702              {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
7703              {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
7704              {'call': 'shmat', 'reason': set([('path', 'dentry')])},
7705              {'call': 'socket', 'reason': set([('path', 'dentry')])},
7706              {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
7707              {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
7708              {'call': 'perf_event_open', 'reason': set([('path', 'dentry')])},
7709              {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
7710              {'call': 'quotactl',
7711               'reason': set([('filename', 'name'), ('path', 'dentry')])},
7712              {'call': 'acct',
7713               'reason': set([('filename', 'name'), ('path', 'dentry')])},
7714              {'call': 'open',
7715               'reason': set([('filename', 'name'), ('path', 'dentry')])},
```



```
7716              {'call': 'unlink', 'reason': set([('filename', 'name')])},
7717              {'call': 'rmdir', 'reason': set([('filename', 'name')])},
7718              {'call': 'dup', 'reason': set([('path', 'dentry')])},
7719              {'call': 'setns', 'reason': set([('path', 'dentry')])},
7720              {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
7721              {'call': 'swapon',
7722               'reason': set([('filename', 'name'), ('path', 'dentry')])},
7723              {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
7724              {'call': 'unlinkat', 'reason': set([('filename', 'name')])},
7725              {'call': 'open_by_handle_at',
7726               'reason': set([('path', 'dentry')])}],
7727  'mq_timedreceive': [{'call': 'msgrcv', 'reason': set([('msg_msg', 'm_ts')])},
7728                      {'call': 'eventfd2',
7729                       'reason': set([('file', 'f_flags'),
7730                                      ('file', 'f_mode'),
7731                                      ('file', 'f_op')])},
7732                      {'call': 'swapoff',
7733                       'reason': set([('file', 'f_flags'),
7734                                      ('file', 'f_mode'),
7735                                      ('file', 'f_op')])},
7736                      {'call': 'memfd_create',
7737                       'reason': set([('file', 'f_flags'),
7738                                      ('file', 'f_mode'),
7739                                      ('file', 'f_op')])},
7740                      {'call': 'remap_file_pages',
7741                       'reason': set([('file', 'f_flags'),
7742                                      ('file', 'f_mode'),
7743                                      ('file', 'f_op')])},
7744                      {'call': 'dup3',
7745                       'reason': set([('file', 'f_flags'),
7746                                      ('file', 'f_mode'),
7747                                      ('file', 'f_op')])},
7748                      {'call': 'epoll_create1',
7749                       'reason': set([('file', 'f_flags'),
7750                                      ('file', 'f_mode'),
7751                                      ('file', 'f_op')])},
7752                      {'call': 'epoll_ctl',
7753                       'reason': set([('file', 'f_flags'),
7754                                      ('file', 'f_mode'),
7755                                      ('file', 'f_op')])},
7756                      {'call': 'flock',
7757                       'reason': set([('file', 'f_flags'),
7758                                      ('file', 'f_mode'),
7759                                      ('file', 'f_op')])},
7760                      {'call': 'openat',
7761                       'reason': set([('file', 'f_flags'),
7762                                      ('file', 'f_mode'),
7763                                      ('file', 'f_op')])},
7764                      {'call': 'uselib',
7765                       'reason': set([('file', 'f_flags'),
7766                                      ('file', 'f_mode'),
7767                                      ('file', 'f_op')])},
7768                      {'call': 'accept4',
7769                       'reason': set([('file', 'f_flags'),
7770                                      ('file', 'f_mode'),
7771                                      ('file', 'f_op')])},
7772                      {'call': 'socketpair',
7773                       'reason': set([('file', 'f_flags'),
7774                                      ('file', 'f_mode'),
7775                                      ('file', 'f_op')])},
7776                      {'call': 'shmat',
7777                       'reason': set([('file', 'f_flags'),
7778                                      ('file', 'f_mode'),
7779                                      ('file', 'f_op')])},
7780                      {'call': 'socket',
7781                       'reason': set([('file', 'f_flags'),
```

```
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'pipe2',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'perf_event_open',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'shmdt',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'acct',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'open',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'mq_getsetattr',
                     'reason': set([('file', 'f_flags'),
                                    ('mq_attr', 'mq_curmsgs'),
                                    ('mq_attr', 'mq_msgsize'),
                                    ('mqueue_inode_info', 'node_cache')])},
                    {'call': 'dup',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'setns',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('mq_attr', 'mq_curmsgs'),
                                    ('mq_attr', 'mq_msgsize'),
                                    ('mqueue_inode_info', 'node_cache'),
                                    ('msg_msg', 'm_ts')])},
                    {'call': 'shmctl',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'swapon',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'mmap_pgoff',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op')])},
                    {'call': 'msgsnd', 'reason': set([('msg_msg', 'm_ts')])},
                    {'call': 'mq_notify',
                     'reason': set([('mq_attr', 'mq_curmsgs'),
                                    ('mq_attr', 'mq_msgsize'),
                                    ('mqueue_inode_info', 'node_cache')])},
                    {'call': 'mq_open',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
                                    ('file', 'f_op'),
                                    ('mq_attr', 'mq_curmsgs'),
                                    ('mq_attr', 'mq_msgsize')])},
                    {'call': 'open_by_handle_at',
                     'reason': set([('file', 'f_flags'),
                                    ('file', 'f_mode'),
```



```
                                    ('file', 'f_op')])}],
 'mq_timedsend': [{'call': 'eventfd2',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'swapoff',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'memfd_create',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'remap_file_pages',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'dup3',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'mq_timedreceive',
                   'reason': set([('mq_attr', 'mq_curmsgs'),
                                  ('mq_attr', 'mq_maxmsg'),
                                  ('mq_attr', 'mq_msgsize'),
                                  ('mqueue_inode_info', 'node_cache')])},
                  {'call': 'epoll_create1',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'epoll_ctl',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'flock',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'openat',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'uselib',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'accept4',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'socketpair',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'shmat',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'socket',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
                  {'call': 'pipe2',
                   'reason': set([('file', 'f_flags'),
                                  ('file', 'f_mode'),
                                  ('file', 'f_op')])},
```

```
7914                 {'call': 'perf_event_open',
7915                  'reason': set([('file', 'f_flags'),
7916                                 ('file', 'f_mode'),
7917                                 ('file', 'f_op')])},
7918                 {'call': 'shmdt',
7919                  'reason': set([('file', 'f_flags'),
7920                                 ('file', 'f_mode'),
7921                                 ('file', 'f_op')])},
7922                 {'call': 'acct',
7923                  'reason': set([('file', 'f_flags'),
7924                                 ('file', 'f_mode'),
7925                                 ('file', 'f_op')])},
7926                 {'call': 'open',
7927                  'reason': set([('file', 'f_flags'),
7928                                 ('file', 'f_mode'),
7929                                 ('file', 'f_op')])},
7930                 {'call': 'mq_getsetattr',
7931                  'reason': set([('file', 'f_flags'),
7932                                 ('mq_attr', 'mq_curmsgs'),
7933                                 ('mq_attr', 'mq_maxmsg'),
7934                                 ('mq_attr', 'mq_msgsize'),
7935                                 ('mqueue_inode_info', 'node_cache')])},
7936                 {'call': 'dup',
7937                  'reason': set([('file', 'f_flags'),
7938                                 ('file', 'f_mode'),
7939                                 ('file', 'f_op')])},
7940                 {'call': 'setns',
7941                  'reason': set([('file', 'f_flags'),
7942                                 ('file', 'f_mode'),
7943                                 ('file', 'f_op')])},
7944                 {'call': 'shmctl',
7945                  'reason': set([('file', 'f_flags'),
7946                                 ('file', 'f_mode'),
7947                                 ('file', 'f_op')])},
7948                 {'call': 'swapon',
7949                  'reason': set([('file', 'f_flags'),
7950                                 ('file', 'f_mode'),
7951                                 ('file', 'f_op')])},
7952                 {'call': 'mmap_pgoff',
7953                  'reason': set([('file', 'f_flags'),
7954                                 ('file', 'f_mode'),
7955                                 ('file', 'f_op')])},
7956                 {'call': 'mq_notify',
7957                  'reason': set([('mq_attr', 'mq_curmsgs'),
7958                                 ('mq_attr', 'mq_maxmsg'),
7959                                 ('mq_attr', 'mq_msgsize'),
7960                                 ('mqueue_inode_info', 'node_cache')])},
7961                 {'call': 'mq_open',
7962                  'reason': set([('file', 'f_flags'),
7963                                 ('file', 'f_mode'),
7964                                 ('file', 'f_op'),
7965                                 ('mq_attr', 'mq_curmsgs'),
7966                                 ('mq_attr', 'mq_maxmsg'),
7967                                 ('mq_attr', 'mq_msgsize')])},
7968                 {'call': 'open_by_handle_at',
7969                  'reason': set([('file', 'f_flags'),
7970                                 ('file', 'f_mode'),
7971                                 ('file', 'f_op')])}],
7972  'mremap': [{'call': 'keyctl',
7973              'reason': set([('task_struct', 'personality')])},
7974             {'call': 'rt_sigtimedwait',
7975              'reason': set([('task_struct', 'personality')])},
7976             {'call': 'msgrcv',
7977              'reason': set([('task_struct', 'personality')])},
7978             {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
7979             {'call': 'swapoff', 'reason': set([('mm_struct', 'map_count')])},
```



```
7980             {'call': 'sched_getaffinity',
7981              'reason': set([('task_struct', 'personality')])},
7982             {'call': 'sched_setparam',
7983              'reason': set([('task_struct', 'personality')])},
7984             {'call': 'ioprio_set',
7985              'reason': set([('task_struct', 'personality')])},
7986             {'call': 'personality',
7987              'reason': set([('task_struct', 'personality')])},
7988             {'call': 'remap_file_pages',
7989              'reason': set([('mm_struct', 'map_count'),
7990                             ('vm_area_struct', 'vm_end'),
7991                             ('vm_area_struct', 'vm_file'),
7992                             ('vm_area_struct', 'vm_flags'),
7993                             ('vm_area_struct', 'vm_next'),
7994                             ('vm_area_struct', 'vm_ops'),
7995                             ('vm_area_struct', 'vm_pgoff'),
7996                             ('vm_area_struct', 'vm_start')])},
7997             {'call': 'io_getevents',
7998              'reason': set([('mm_struct', 'map_count')])},
7999             {'call': 'getppid',
8000              'reason': set([('task_struct', 'personality')])},
8001             {'call': 'mq_timedreceive',
8002              'reason': set([('task_struct', 'personality')])},
8003             {'call': 'capget',
8004              'reason': set([('task_struct', 'personality')])},
8005             {'call': 'sched_setaffinity',
8006              'reason': set([('task_struct', 'personality')])},
8007             {'call': 'signal',
8008              'reason': set([('task_struct', 'personality')])},
8009             {'call': 'semtimedop',
8010              'reason': set([('task_struct', 'personality')])},
8011             {'call': 'umount',
8012              'reason': set([('task_struct', 'personality')])},
8013             {'call': 'sched_rr_get_interval',
8014              'reason': set([('task_struct', 'personality')])},
8015             {'call': 'rt_sigprocmask',
8016              'reason': set([('task_struct', 'personality')])},
8017             {'call': 'setsid',
8018              'reason': set([('task_struct', 'personality')])},
8019             {'call': 'sigaltstack',
8020              'reason': set([('task_struct', 'personality')])},
8021             {'call': 'sched_setattr',
8022              'reason': set([('task_struct', 'personality')])},
8023             {'call': 'migrate_pages',
8024              'reason': set([('mm_struct', 'map_count'),
8025                             ('task_struct', 'personality')])},
8026             {'call': 'getitimer',
8027              'reason': set([('task_struct', 'personality')])},
8028             {'call': 'setpgid',
8029              'reason': set([('task_struct', 'personality')])},
8030             {'call': 'getsid',
8031              'reason': set([('task_struct', 'personality')])},
8032             {'call': 'prlimit64',
8033              'reason': set([('task_struct', 'personality')])},
8034             {'call': 'perf_event_open',
8035              'reason': set([('task_struct', 'personality')])},
8036             {'call': 'shmdt',
8037              'reason': set([('mm_struct', 'map_count'),
8038                             ('vm_area_struct', 'vm_end'),
8039                             ('vm_area_struct', 'vm_file'),
8040                             ('vm_area_struct', 'vm_flags'),
8041                             ('vm_area_struct', 'vm_next'),
8042                             ('vm_area_struct', 'vm_ops'),
8043                             ('vm_area_struct', 'vm_pgoff'),
8044                             ('vm_area_struct', 'vm_start')])},
8045             {'call': 'rt_sigaction',
```

```
8046             'reason': set([('task_struct', 'personality')])},
8047            {'call': 'getpgid',
8048             'reason': set([('task_struct', 'personality')])},
8049            {'call': 'brk',
8050             'reason': set([('mm_struct', 'map_count'),
8051                            ('vm_area_struct', 'vm_end'),
8052                            ('vm_area_struct', 'vm_file'),
8053                            ('vm_area_struct', 'vm_flags'),
8054                            ('vm_area_struct', 'vm_next'),
8055                            ('vm_area_struct', 'vm_ops'),
8056                            ('vm_area_struct', 'vm_pgoff'),
8057                            ('vm_area_struct', 'vm_start')])},
8058            {'call': 'getpriority',
8059             'reason': set([('task_struct', 'personality')])},
8060            {'call': 'sigaction',
8061             'reason': set([('task_struct', 'personality')])},
8062            {'call': 'setns',
8063             'reason': set([('task_struct', 'personality')])},
8064            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
8065            {'call': 'get_mempolicy',
8066             'reason': set([('mm_struct', 'map_count'),
8067                            ('vm_area_struct', 'vm_end'),
8068                            ('vm_area_struct', 'vm_file'),
8069                            ('vm_area_struct', 'vm_flags'),
8070                            ('vm_area_struct', 'vm_next'),
8071                            ('vm_area_struct', 'vm_ops'),
8072                            ('vm_area_struct', 'vm_pgoff'),
8073                            ('vm_area_struct', 'vm_start')])},
8074            {'call': 'get_robust_list',
8075             'reason': set([('task_struct', 'personality')])},
8076            {'call': 'mq_timedsend',
8077             'reason': set([('task_struct', 'personality')])},
8078            {'call': 'sched_getscheduler',
8079             'reason': set([('task_struct', 'personality')])},
8080            {'call': 'ptrace',
8081             'reason': set([('task_struct', 'personality')])},
8082            {'call': 'munlockall',
8083             'reason': set([('vm_area_struct', 'vm_end'),
8084                            ('vm_area_struct', 'vm_file'),
8085                            ('vm_area_struct', 'vm_flags'),
8086                            ('vm_area_struct', 'vm_next'),
8087                            ('vm_area_struct', 'vm_ops'),
8088                            ('vm_area_struct', 'vm_pgoff'),
8089                            ('vm_area_struct', 'vm_start')])},
8090            {'call': 'pkey_mprotect',
8091             'reason': set([('vm_area_struct', 'vm_end'),
8092                            ('vm_area_struct', 'vm_file'),
8093                            ('vm_area_struct', 'vm_flags'),
8094                            ('vm_area_struct', 'vm_next'),
8095                            ('vm_area_struct', 'vm_ops'),
8096                            ('vm_area_struct', 'vm_pgoff'),
8097                            ('vm_area_struct', 'vm_start')])},
8098            {'call': 'madvise',
8099             'reason': set([('vm_area_struct', 'vm_end'),
8100                            ('vm_area_struct', 'vm_file'),
8101                            ('vm_area_struct', 'vm_flags'),
8102                            ('vm_area_struct', 'vm_next'),
8103                            ('vm_area_struct', 'vm_ops'),
8104                            ('vm_area_struct', 'vm_pgoff'),
8105                            ('vm_area_struct', 'vm_start')])},
8106            {'call': 'sched_getattr',
8107             'reason': set([('task_struct', 'personality')])},
8108            {'call': 'getrusage',
8109             'reason': set([('mm_struct', 'map_count'),
8110                            ('task_struct', 'personality')])},
8111            {'call': 'sched_setscheduler',
```



```
8112             'reason': set([('task_struct', 'personality')])},
8113            {'call': 'setitimer',
8114             'reason': set([('task_struct', 'personality')])},
8115            {'call': 'ioprio_get',
8116             'reason': set([('task_struct', 'personality')])},
8117            {'call': 'vfork',
8118             'reason': set([('task_struct', 'personality')])},
8119            {'call': 'io_setup', 'reason': set([('mm_struct', 'map_count')])},
8120            {'call': 'mprotect',
8121             'reason': set([('vm_area_struct', 'vm_end'),
8122                            ('vm_area_struct', 'vm_file'),
8123                            ('vm_area_struct', 'vm_flags'),
8124                            ('vm_area_struct', 'vm_next'),
8125                            ('vm_area_struct', 'vm_ops'),
8126                            ('vm_area_struct', 'vm_pgoff'),
8127                            ('vm_area_struct', 'vm_start')])},
8128            {'call': 'io_destroy',
8129             'reason': set([('mm_struct', 'map_count')])},
8130            {'call': 'mbind', 'reason': set([('mm_struct', 'map_count')])},
8131            {'call': 'prctl',
8132             'reason': set([('mm_struct', 'map_count'),
8133                            ('task_struct', 'personality'),
8134                            ('vm_area_struct', 'vm_end'),
8135                            ('vm_area_struct', 'vm_file'),
8136                            ('vm_area_struct', 'vm_flags'),
8137                            ('vm_area_struct', 'vm_next'),
8138                            ('vm_area_struct', 'vm_ops'),
8139                            ('vm_area_struct', 'vm_pgoff'),
8140                            ('vm_area_struct', 'vm_start')])},
8141            {'call': 'move_pages',
8142             'reason': set([('mm_struct', 'map_count'),
8143                            ('task_struct', 'personality')])},
8144            {'call': 'modify_ldt',
8145             'reason': set([('mm_struct', 'map_count')])},
8146            {'call': 'munlock',
8147             'reason': set([('vm_area_struct', 'vm_end'),
8148                            ('vm_area_struct', 'vm_file'),
8149                            ('vm_area_struct', 'vm_flags'),
8150                            ('vm_area_struct', 'vm_next'),
8151                            ('vm_area_struct', 'vm_ops'),
8152                            ('vm_area_struct', 'vm_pgoff'),
8153                            ('vm_area_struct', 'vm_start')])},
8154            {'call': 'setpriority',
8155             'reason': set([('task_struct', 'personality')])},
8156            {'call': 'mincore',
8157             'reason': set([('mm_struct', 'map_count'),
8158                            ('vm_area_struct', 'vm_end'),
8159                            ('vm_area_struct', 'vm_file'),
8160                            ('vm_area_struct', 'vm_flags'),
8161                            ('vm_area_struct', 'vm_next'),
8162                            ('vm_area_struct', 'vm_ops'),
8163                            ('vm_area_struct', 'vm_pgoff'),
8164                            ('vm_area_struct', 'vm_start')])},
8165            {'call': 'clone',
8166             'reason': set([('task_struct', 'personality')])},
8167            {'call': 'sched_getparam',
8168             'reason': set([('task_struct', 'personality')])},
8169            {'call': 'io_cancel',
8170             'reason': set([('mm_struct', 'map_count')])},
8171            {'call': 'mlockall',
8172             'reason': set([('vm_area_struct', 'vm_end'),
8173                            ('vm_area_struct', 'vm_file'),
8174                            ('vm_area_struct', 'vm_flags'),
8175                            ('vm_area_struct', 'vm_next'),
8176                            ('vm_area_struct', 'vm_ops'),
8177                            ('vm_area_struct', 'vm_pgoff'),
```

```
8178                          (’vm_area_struct’, ’vm_start’)])}],
8179  ’msgctl’: [{’call’: ’keyctl’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8180              {’call’: ’rt_sigtimedwait’,
8181               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8182              {’call’: ’msgrcv’,
8183               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’),
8184                              (’mm_segment_t’, ’seg’)])},
8185              {’call’: ’mq_unlink’,
8186               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8187              {’call’: ’kill’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8188              {’call’: ’msgget’,
8189               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8190              {’call’: ’sched_getaffinity’,
8191               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8192              {’call’: ’sched_setparam’,
8193               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8194              {’call’: ’ioprio_set’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8195              {’call’: ’getppid’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8196              {’call’: ’ioperm’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8197              {’call’: ’mq_timedreceive’,
8198               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8199              {’call’: ’capget’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8200              {’call’: ’sched_setaffinity’,
8201               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8202              {’call’: ’signal’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8203              {’call’: ’semtimedop’,
8204               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’),
8205                              (’mm_segment_t’, ’seg’)])},
8206              {’call’: ’umount’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8207              {’call’: ’sched_rr_get_interval’,
8208               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8209              {’call’: ’semctl’,
8210               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8211              {’call’: ’shmget’,
8212               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8213              {’call’: ’rt_sigprocmask’,
8214               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8215              {’call’: ’setsid’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8216              {’call’: ’sigaltstack’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8217              {’call’: ’sched_setattr’,
8218               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8219              {’call’: ’migrate_pages’,
8220               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8221              {’call’: ’getitimer’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8222              {’call’: ’setpgid’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8223              {’call’: ’semget’,
8224               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8225              {’call’: ’getsid’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8226              {’call’: ’shmat’,
8227               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8228              {’call’: ’prlimit64’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8229              {’call’: ’perf_event_open’,
8230               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8231              {’call’: ’rt_sigaction’,
8232               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8233              {’call’: ’getpgid’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8234              {’call’: ’getpriority’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8235              {’call’: ’sigaction’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8236              {’call’: ’setns’,
8237               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’),
8238                              (’mm_segment_t’, ’seg’)])},
8239              {’call’: ’fork’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8240              {’call’: ’get_robust_list’,
8241               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8242              {’call’: ’mq_timedsend’,
8243               ’reason’: set([(’mm_segment_t’, ’seg’)])},
```



```
8244              {’call’: ’sched_getscheduler’,
8245               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8246              {’call’: ’ptrace’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8247              {’call’: ’shmctl’,
8248               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8249              {’call’: ’sched_getattr’,
8250               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8251              {’call’: ’getrusage’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8252              {’call’: ’sched_setscheduler’,
8253               ’reason’: set([(’mm_segment_t’, ’seg’)])},
8254              {’call’: ’setitimer’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8255              {’call’: ’ioprio_get’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8256              {’call’: ’vfork’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8257              {’call’: ’prctl’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8258              {’call’: ’move_pages’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8259              {’call’: ’msgsnd’,
8260               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8261              {’call’: ’setpriority’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8262              {’call’: ’clone’, ’reason’: set([(’mm_segment_t’, ’seg’)])},
8263              {’call’: ’mq_open’,
8264               ’reason’: set([(’ipc_namespace’, ’msg_ctlmnb’)])},
8265              {’call’: ’sched_getparam’,
8266               ’reason’: set([(’mm_segment_t’, ’seg’)])}],
8267  ’msgrcv’: [{’call’: ’mq_timedreceive’, ’reason’: set([(’msg_msg’, ’m_ts’)])},
8268              {’call’: ’mq_timedsend’, ’reason’: set([(’msg_msg’, ’m_ts’)])},
8269              {’call’: ’msgsnd’, ’reason’: set([(’msg_msg’, ’m_ts’)])}],
8270  ’msgsnd’: [{’call’: ’msgrcv’,
8271              ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8272              {’call’: ’mq_unlink’,
8273               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8274              {’call’: ’msgget’,
8275               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8276              {’call’: ’semtimedop’,
8277               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8278              {’call’: ’semctl’,
8279               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8280              {’call’: ’shmget’,
8281               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8282              {’call’: ’msgctl’,
8283               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8284              {’call’: ’semget’,
8285               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8286              {’call’: ’shmat’,
8287               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8288              {’call’: ’setns’,
8289               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8290              {’call’: ’shmctl’,
8291               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])},
8292              {’call’: ’mq_open’,
8293               ’reason’: set([(’ipc_namespace’, ’msg_ctlmax’)])}],
8294  ’msync’: [{’call’: ’remap_file_pages’,
8295             ’reason’: set([(’vm_area_struct’, ’vm_file’),
8296                            (’vm_area_struct’, ’vm_flags’)])},
8297            {’call’: ’shmdt’,
8298             ’reason’: set([(’vm_area_struct’, ’vm_file’),
8299                            (’vm_area_struct’, ’vm_flags’)])},
8300            {’call’: ’brk’,
8301             ’reason’: set([(’vm_area_struct’, ’vm_file’),
8302                            (’vm_area_struct’, ’vm_flags’)])},
8303            {’call’: ’get_mempolicy’,
8304             ’reason’: set([(’vm_area_struct’, ’vm_file’),
8305                            (’vm_area_struct’, ’vm_flags’)])},
8306            {’call’: ’munlockall’,
8307             ’reason’: set([(’vm_area_struct’, ’vm_file’),
8308                            (’vm_area_struct’, ’vm_flags’)])},
8309            {’call’: ’pkey_mprotect’,
```

```
8310              'reason': set([('vm_area_struct', 'vm_file'),
8311                             ('vm_area_struct', 'vm_flags')])},
8312             {'call': 'madvise',
8313              'reason': set([('vm_area_struct', 'vm_file'),
8314                             ('vm_area_struct', 'vm_flags')])},
8315             {'call': 'mprotect',
8316              'reason': set([('vm_area_struct', 'vm_file'),
8317                             ('vm_area_struct', 'vm_flags')])},
8318             {'call': 'mremap',
8319              'reason': set([('vm_area_struct', 'vm_file'),
8320                             ('vm_area_struct', 'vm_flags')])},
8321             {'call': 'prctl',
8322              'reason': set([('vm_area_struct', 'vm_file'),
8323                             ('vm_area_struct', 'vm_flags')])},
8324             {'call': 'munlock',
8325              'reason': set([('vm_area_struct', 'vm_file'),
8326                             ('vm_area_struct', 'vm_flags')])},
8327             {'call': 'mincore',
8328              'reason': set([('vm_area_struct', 'vm_file'),
8329                             ('vm_area_struct', 'vm_flags')])},
8330             {'call': 'mlockall',
8331              'reason': set([('vm_area_struct', 'vm_file'),
8332                             ('vm_area_struct', 'vm_flags')])}],
8333  'munlock': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
8334              {'call': 'rt_sigtimedwait',
8335               'reason': set([('task_struct', 'mm')])},
8336              {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
8337              {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
8338              {'call': 'sched_getaffinity',
8339               'reason': set([('task_struct', 'mm')])},
8340              {'call': 'sched_setparam',
8341               'reason': set([('task_struct', 'mm')])},
8342              {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
8343              {'call': 'remap_file_pages',
8344               'reason': set([('vm_area_struct', 'vm_end'),
8345                              ('vm_area_struct', 'vm_start')])},
8346              {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
8347              {'call': 'mq_timedreceive',
8348               'reason': set([('task_struct', 'mm')])},
8349              {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
8350              {'call': 'sched_setaffinity',
8351               'reason': set([('task_struct', 'mm')])},
8352              {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
8353              {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
8354              {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
8355              {'call': 'sched_rr_get_interval',
8356               'reason': set([('task_struct', 'mm')])},
8357              {'call': 'rt_sigprocmask',
8358               'reason': set([('task_struct', 'mm')])},
8359              {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
8360              {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
8361              {'call': 'sched_setattr',
8362               'reason': set([('task_struct', 'mm')])},
8363              {'call': 'migrate_pages',
8364               'reason': set([('task_struct', 'mm')])},
8365              {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
8366              {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
8367              {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
8368              {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
8369              {'call': 'perf_event_open',
8370               'reason': set([('task_struct', 'mm')])},
8371              {'call': 'shmdt',
8372               'reason': set([('vm_area_struct', 'vm_end'),
8373                              ('vm_area_struct', 'vm_start')])},
8374              {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
8375              {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
```



```
8376              {'call': 'brk',
8377               'reason': set([('vm_area_struct', 'vm_end'),
8378                              ('vm_area_struct', 'vm_start')])},
8379              {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
8380              {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
8381              {'call': 'setns', 'reason': set([('task_struct', 'mm')])},
8382              {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
8383              {'call': 'get_mempolicy',
8384               'reason': set([('vm_area_struct', 'vm_end'),
8385                              ('vm_area_struct', 'vm_start')])},
8386              {'call': 'get_robust_list',
8387               'reason': set([('task_struct', 'mm')])},
8388              {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
8389              {'call': 'sched_getscheduler',
8390               'reason': set([('task_struct', 'mm')])},
8391              {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
8392              {'call': 'munlockall',
8393               'reason': set([('vm_area_struct', 'vm_end'),
8394                              ('vm_area_struct', 'vm_start')])},
8395              {'call': 'pkey_mprotect',
8396               'reason': set([('vm_area_struct', 'vm_end'),
8397                              ('vm_area_struct', 'vm_start')])},
8398              {'call': 'madvise',
8399               'reason': set([('vm_area_struct', 'vm_end'),
8400                              ('vm_area_struct', 'vm_start')])},
8401              {'call': 'sched_getattr',
8402               'reason': set([('task_struct', 'mm')])},
8403              {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
8404              {'call': 'sched_setscheduler',
8405               'reason': set([('task_struct', 'mm')])},
8406              {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
8407              {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
8408              {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
8409              {'call': 'mprotect',
8410               'reason': set([('vm_area_struct', 'vm_end'),
8411                              ('vm_area_struct', 'vm_start')])},
8412              {'call': 'mremap',
8413               'reason': set([('vm_area_struct', 'vm_end'),
8414                              ('vm_area_struct', 'vm_start')])},
8415              {'call': 'prctl',
8416               'reason': set([('task_struct', 'mm'),
8417                              ('vm_area_struct', 'vm_end'),
8418                              ('vm_area_struct', 'vm_start')])},
8419              {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
8420              {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
8421              {'call': 'mincore',
8422               'reason': set([('vm_area_struct', 'vm_end'),
8423                              ('vm_area_struct', 'vm_start')])},
8424              {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
8425              {'call': 'sched_getparam',
8426               'reason': set([('task_struct', 'mm')])},
8427              {'call': 'mlockall',
8428               'reason': set([('vm_area_struct', 'vm_end'),
8429                              ('vm_area_struct', 'vm_start')])}],
8430  'munlockall': [{'call': 'remap_file_pages',
8431                  'reason': set([('vm_area_struct', 'vm_end'),
8432                                 ('vm_area_struct', 'vm_start')])},
8433                 {'call': 'shmdt',
8434                  'reason': set([('vm_area_struct', 'vm_end'),
8435                                 ('vm_area_struct', 'vm_start')])},
8436                 {'call': 'brk',
8437                  'reason': set([('vm_area_struct', 'vm_end'),
8438                                 ('vm_area_struct', 'vm_start')])},
8439                 {'call': 'get_mempolicy',
8440                  'reason': set([('vm_area_struct', 'vm_end'),
8441                                 ('vm_area_struct', 'vm_start')])},
```

```
8442                 {'call': 'pkey_mprotect',
8443                  'reason': set([('vm_area_struct', 'vm_end'),
8444                                 ('vm_area_struct', 'vm_start')])},
8445                 {'call': 'madvise',
8446                  'reason': set([('vm_area_struct', 'vm_end'),
8447                                 ('vm_area_struct', 'vm_start')])},
8448                 {'call': 'mprotect',
8449                  'reason': set([('vm_area_struct', 'vm_end'),
8450                                 ('vm_area_struct', 'vm_start')])},
8451                 {'call': 'mremap',
8452                  'reason': set([('vm_area_struct', 'vm_end'),
8453                                 ('vm_area_struct', 'vm_start')])},
8454                 {'call': 'prctl',
8455                  'reason': set([('vm_area_struct', 'vm_end'),
8456                                 ('vm_area_struct', 'vm_start')])},
8457                 {'call': 'munlock',
8458                  'reason': set([('vm_area_struct', 'vm_end'),
8459                                 ('vm_area_struct', 'vm_start')])},
8460                 {'call': 'mincore',
8461                  'reason': set([('vm_area_struct', 'vm_end'),
8462                                 ('vm_area_struct', 'vm_start')])},
8463                 {'call': 'mlockall',
8464                  'reason': set([('vm_area_struct', 'vm_end'),
8465                                 ('vm_area_struct', 'vm_start')])}],
8466  'name_to_handle_at': [{'call': 'syncfs',
8467                         'reason': set([('super_block', 's_export_op')])},
8468                        {'call': 'eventfd2',
8469                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8470                        {'call': 'swapoff',
8471                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8472                        {'call': 'pivot_root',
8473                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8474                        {'call': 'memfd_create',
8475                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8476                        {'call': 'remap_file_pages',
8477                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8478                        {'call': 'dup3',
8479                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8480                        {'call': 'ustat',
8481                         'reason': set([('super_block', 's_export_op')])},
8482                        {'call': 'unshare',
8483                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8484                        {'call': 'umount',
8485                         'reason': set([('super_block', 's_export_op')])},
8486                        {'call': 'epoll_create1',
8487                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8488                        {'call': 'epoll_ctl',
8489                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8490                        {'call': 'flock',
8491                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8492                        {'call': 'openat',
8493                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8494                        {'call': 'lookup_dcookie',
8495                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8496                        {'call': 'uselib',
8497                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8498                        {'call': 'accept4',
8499                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8500                        {'call': 'socketpair',
8501                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8502                        {'call': 'getcwd',
8503                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8504                        {'call': 'shmat',
8505                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8506                        {'call': 'socket',
8507                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```



```
8508                        {'call': 'pipe2',
8509                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8510                        {'call': 'perf_event_open',
8511                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8512                        {'call': 'shmdt',
8513                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8514                        {'call': 'quotactl',
8515                         'reason': set([('path', 'dentry'),
8516                                        ('path', 'mnt'),
8517                                        ('super_block', 's_export_op')])},
8518                        {'call': 'acct',
8519                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8520                        {'call': 'open',
8521                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8522                        {'call': 'dup',
8523                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8524                        {'call': 'setns',
8525                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8526                        {'call': 'shmctl',
8527                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8528                        {'call': 'swapon',
8529                         'reason': set([('path', 'dentry'),
8530                                        ('path', 'mnt'),
8531                                        ('super_block', 's_export_op')])},
8532                        {'call': 'mmap_pgoff',
8533                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8534                        {'call': 'mq_open',
8535                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
8536                        {'call': 'open_by_handle_at',
8537                         'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
8538  'nanosleep': [{'call': 'rt_sigtimedwait',
8539                 'reason': set([('timespec', 'tv_nsec'),
8540                                ('timespec', 'tv_sec')])},
8541                {'call': 'mq_unlink',
8542                 'reason': set([('timespec', 'tv_nsec'),
8543                                ('timespec', 'tv_sec')])},
8544                {'call': 'swapoff',
8545                 'reason': set([('timespec', 'tv_nsec'),
8546                                ('timespec', 'tv_sec')])},
8547                {'call': 'fchmod',
8548                 'reason': set([('timespec', 'tv_nsec'),
8549                                ('timespec', 'tv_sec')])},
8550                {'call': 'memfd_create',
8551                 'reason': set([('timespec', 'tv_nsec'),
8552                                ('timespec', 'tv_sec')])},
8553                {'call': 'readlinkat',
8554                 'reason': set([('timespec', 'tv_nsec'),
8555                                ('timespec', 'tv_sec')])},
8556                {'call': 'io_getevents',
8557                 'reason': set([('timespec', 'tv_nsec'),
8558                                ('timespec', 'tv_sec')])},
8559                {'call': 'fchown',
8560                 'reason': set([('timespec', 'tv_nsec'),
8561                                ('timespec', 'tv_sec')])},
8562                {'call': 'mq_timedreceive',
8563                 'reason': set([('timespec', 'tv_nsec'),
8564                                ('timespec', 'tv_sec')])},
8565                {'call': 'utime',
8566                 'reason': set([('timespec', 'tv_nsec'),
8567                                ('timespec', 'tv_sec')])},
8568                {'call': 'semtimedop',
8569                 'reason': set([('timespec', 'tv_nsec'),
8570                                ('timespec', 'tv_sec')])},
8571                {'call': 'settimeofday',
8572                 'reason': set([('timespec', 'tv_nsec'),
8573                                ('timespec', 'tv_sec')])},
```

```
8574                {'call': 'sched_rr_get_interval',
8575                 'reason': set([('timespec', 'tv_nsec'),
8576                                ('timespec', 'tv_sec')])},
8577                {'call': 'timerfd_gettime',
8578                 'reason': set([('timespec', 'tv_nsec'),
8579                                ('timespec', 'tv_sec')])},
8580                {'call': 'pselect6',
8581                 'reason': set([('timespec', 'tv_nsec'),
8582                                ('timespec', 'tv_sec')])},
8583                {'call': 'uselib',
8584                 'reason': set([('timespec', 'tv_nsec'),
8585                                ('timespec', 'tv_sec')])},
8586                {'call': 'fchmodat',
8587                 'reason': set([('timespec', 'tv_nsec'),
8588                                ('timespec', 'tv_sec')])},
8589                {'call': 'inotify_add_watch',
8590                 'reason': set([('timespec', 'tv_nsec'),
8591                                ('timespec', 'tv_sec')])},
8592                {'call': 'timer_settime',
8593                 'reason': set([('timespec', 'tv_nsec'),
8594                                ('timespec', 'tv_sec')])},
8595                {'call': 'ftruncate',
8596                 'reason': set([('timespec', 'tv_nsec'),
8597                                ('timespec', 'tv_sec')])},
8598                {'call': 'timer_gettime',
8599                 'reason': set([('timespec', 'tv_nsec'),
8600                                ('timespec', 'tv_sec')])},
8601                {'call': 'ioctl',
8602                 'reason': set([('timespec', 'tv_nsec'),
8603                                ('timespec', 'tv_sec')])},
8604                {'call': 'linkat',
8605                 'reason': set([('timespec', 'tv_nsec'),
8606                                ('timespec', 'tv_sec')])},
8607                {'call': 'stime',
8608                 'reason': set([('timespec', 'tv_nsec'),
8609                                ('timespec', 'tv_sec')])},
8610                {'call': 'futimesat',
8611                 'reason': set([('timespec', 'tv_nsec'),
8612                                ('timespec', 'tv_sec')])},
8613                {'call': 'poll',
8614                 'reason': set([('timespec', 'tv_nsec'),
8615                                ('timespec', 'tv_sec')])},
8616                {'call': 'select',
8617                 'reason': set([('timespec', 'tv_nsec'),
8618                                ('timespec', 'tv_sec')])},
8619                {'call': 'unlink',
8620                 'reason': set([('timespec', 'tv_nsec'),
8621                                ('timespec', 'tv_sec')])},
8622                {'call': 'mq_getsetattr',
8623                 'reason': set([('timespec', 'tv_nsec'),
8624                                ('timespec', 'tv_sec')])},
8625                {'call': 'faccessat',
8626                 'reason': set([('timespec', 'tv_nsec'),
8627                                ('timespec', 'tv_sec')])},
8628                {'call': 'mq_timedsend',
8629                 'reason': set([('timespec', 'tv_nsec'),
8630                                ('timespec', 'tv_sec')])},
8631                {'call': 'swapon',
8632                 'reason': set([('timespec', 'tv_nsec'),
8633                                ('timespec', 'tv_sec')])},
8634                {'call': 'epoll_wait',
8635                 'reason': set([('timespec', 'tv_nsec'),
8636                                ('timespec', 'tv_sec')])},
8637                {'call': 'fchownat',
8638                 'reason': set([('timespec', 'tv_nsec'),
8639                                ('timespec', 'tv_sec')])},
```



```
8640                {'call': 'fstat',
8641                 'reason': set([('timespec', 'tv_nsec'),
8642                                ('timespec', 'tv_sec')])},
8643                {'call': 'timerfd_settime',
8644                 'reason': set([('timespec', 'tv_nsec'),
8645                                ('timespec', 'tv_sec')])},
8646                {'call': 'mq_notify',
8647                 'reason': set([('timespec', 'tv_nsec'),
8648                                ('timespec', 'tv_sec')])},
8649                {'call': 'sendfile',
8650                 'reason': set([('timespec', 'tv_nsec'),
8651                                ('timespec', 'tv_sec')])},
8652                {'call': 'newfstat',
8653                 'reason': set([('timespec', 'tv_nsec'),
8654                                ('timespec', 'tv_sec')])},
8655                {'call': 'clock_nanosleep',
8656                 'reason': set([('timespec', 'tv_nsec'),
8657                                ('timespec', 'tv_sec')])},
8658                {'call': 'unlinkat',
8659                 'reason': set([('timespec', 'tv_nsec'),
8660                                ('timespec', 'tv_sec')])},
8661                {'call': 'futex',
8662                 'reason': set([('timespec', 'tv_nsec'),
8663                                ('timespec', 'tv_sec')])},
8664                {'call': 'recvmmsg',
8665                 'reason': set([('timespec', 'tv_nsec'),
8666                                ('timespec', 'tv_sec')])},
8667                {'call': 'sendfile64',
8668                 'reason': set([('timespec', 'tv_nsec'),
8669                                ('timespec', 'tv_sec')])},
8670                {'call': 'ppoll',
8671                 'reason': set([('timespec', 'tv_nsec'),
8672                                ('timespec', 'tv_sec')])}],
8673  'newfstat': [{'call': 'newlstat',
8674                'reason': set([('compat_stat', 'st_ino'),
8675                               ('compat_stat', 'st_nlink'),
8676                               ('stat', 'st_ino'),
8677                               ('stat', 'st_nlink')])},
8678               {'call': 'newfstatat',
8679                'reason': set([('compat_stat', 'st_ino'),
8680                               ('compat_stat', 'st_nlink'),
8681                               ('stat', 'st_ino'),
8682                               ('stat', 'st_nlink')])},
8683               {'call': 'newstat',
8684                'reason': set([('compat_stat', 'st_ino'),
8685                               ('compat_stat', 'st_nlink'),
8686                               ('stat', 'st_ino'),
8687                               ('stat', 'st_nlink')])},
8688               {'call': 'fstat',
8689                'reason': set([('kstat', 'dev'),
8690                               ('kstat', 'ino'),
8691                               ('kstat', 'nlink'),
8692                               ('kstat', 'rdev'),
8693                               ('kstat', 'size')])}],
8694  'newfstatat': [{'call': 'newlstat',
8695                  'reason': set([('compat_stat', 'st_ino'),
8696                                 ('compat_stat', 'st_nlink'),
8697                                 ('stat', 'st_ino'),
8698                                 ('stat', 'st_nlink')])},
8699                 {'call': 'newstat',
8700                  'reason': set([('compat_stat', 'st_ino'),
8701                                 ('compat_stat', 'st_nlink'),
8702                                 ('stat', 'st_ino'),
8703                                 ('stat', 'st_nlink')])},
8704                 {'call': 'fstat',
8705                  'reason': set([('kstat', 'dev'),
```

```
                                ('kstat', 'ino'),
                                ('kstat', 'nlink'),
                                ('kstat', 'rdev'),
                                ('kstat', 'size')])},
                {'call': 'newfstat',
                 'reason': set([('compat_stat', 'st_ino'),
                                ('compat_stat', 'st_nlink'),
                                ('kstat', 'dev'),
                                ('kstat', 'ino'),
                                ('kstat', 'nlink'),
                                ('kstat', 'rdev'),
                                ('kstat', 'size'),
                                ('stat', 'st_ino'),
                                ('stat', 'st_nlink')])}],
 'newlstat': [{'call': 'newfstatat',
               'reason': set([('compat_stat', 'st_ino'),
                              ('compat_stat', 'st_nlink'),
                              ('stat', 'st_ino'),
                              ('stat', 'st_nlink')])},
              {'call': 'newstat',
               'reason': set([('compat_stat', 'st_ino'),
                              ('compat_stat', 'st_nlink'),
                              ('stat', 'st_ino'),
                              ('stat', 'st_nlink')])},
              {'call': 'fstat',
               'reason': set([('kstat', 'dev'),
                              ('kstat', 'ino'),
                              ('kstat', 'nlink'),
                              ('kstat', 'rdev'),
                              ('kstat', 'size')])},
              {'call': 'newfstat',
               'reason': set([('compat_stat', 'st_ino'),
                              ('compat_stat', 'st_nlink'),
                              ('kstat', 'dev'),
                              ('kstat', 'ino'),
                              ('kstat', 'nlink'),
                              ('kstat', 'rdev'),
                              ('kstat', 'size'),
                              ('stat', 'st_ino'),
                              ('stat', 'st_nlink')])}],
 'newstat': [{'call': 'newlstat',
              'reason': set([('compat_stat', 'st_ino'),
                             ('compat_stat', 'st_nlink'),
                             ('stat', 'st_ino'),
                             ('stat', 'st_nlink')])},
             {'call': 'newfstatat',
              'reason': set([('compat_stat', 'st_ino'),
                             ('compat_stat', 'st_nlink'),
                             ('stat', 'st_ino'),
                             ('stat', 'st_nlink')])},
             {'call': 'fstat',
              'reason': set([('kstat', 'dev'),
                             ('kstat', 'ino'),
                             ('kstat', 'nlink'),
                             ('kstat', 'rdev'),
                             ('kstat', 'size')])},
             {'call': 'newfstat',
              'reason': set([('compat_stat', 'st_ino'),
                             ('compat_stat', 'st_nlink'),
                             ('kstat', 'dev'),
                             ('kstat', 'ino'),
                             ('kstat', 'nlink'),
                             ('kstat', 'rdev'),
                             ('kstat', 'size'),
                             ('stat', 'st_ino'),
                             ('stat', 'st_nlink')])}],
```



```
 'newuname': [{'call': 'keyctl',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'msgrcv',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'kill',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sched_getaffinity',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sched_setparam',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'ioprio_set',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'personality',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'getppid',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'mq_timedreceive',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'capget',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sched_setaffinity',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'signal',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'semtimedop',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'umount',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'setsid',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sigaltstack',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sched_setattr',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'migrate_pages',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'getitimer',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'setpgid',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'getsid',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'prlimit64',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'perf_event_open',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'rt_sigaction',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'getpgid',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'getpriority',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'sigaction',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'setns',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'fork',
               'reason': set([('task_struct', 'personality')])},
              {'call': 'get_robust_list',
               'reason': set([('task_struct', 'personality')])},
```

```
8838              {'call': 'mq_timedsend',
8839               'reason': set([('task_struct', 'personality')])},
8840              {'call': 'sched_getscheduler',
8841               'reason': set([('task_struct', 'personality')])},
8842              {'call': 'ptrace',
8843               'reason': set([('task_struct', 'personality')])},
8844              {'call': 'sched_getattr',
8845               'reason': set([('task_struct', 'personality')])},
8846              {'call': 'getrusage',
8847               'reason': set([('task_struct', 'personality')])},
8848              {'call': 'sched_setscheduler',
8849               'reason': set([('task_struct', 'personality')])},
8850              {'call': 'setitimer',
8851               'reason': set([('task_struct', 'personality')])},
8852              {'call': 'ioprio_get',
8853               'reason': set([('task_struct', 'personality')])},
8854              {'call': 'vfork',
8855               'reason': set([('task_struct', 'personality')])},
8856              {'call': 'prctl',
8857               'reason': set([('task_struct', 'personality')])},
8858              {'call': 'move_pages',
8859               'reason': set([('task_struct', 'personality')])},
8860              {'call': 'setpriority',
8861               'reason': set([('task_struct', 'personality')])},
8862              {'call': 'clone',
8863               'reason': set([('task_struct', 'personality')])},
8864              {'call': 'sched_getparam',
8865               'reason': set([('task_struct', 'personality')])}],
8866  'old_getrlimit': [{'call': 'setrlimit',
8867                     'reason': set([('rlimit', 'rlim_cur'),
8868                                    ('rlimit', 'rlim_max')])},
8869                    {'call': 'prlimit64',
8870                     'reason': set([('rlimit', 'rlim_cur'),
8871                                    ('rlimit', 'rlim_max')])}],
8872  'old_readdir': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
8873                  {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
8874                  {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
8875                  {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
8876                  {'call': 'readahead', 'reason': set([('fd', 'flags')])},
8877                  {'call': 'getdents', 'reason': set([('fd', 'flags')])},
8878                  {'call': 'writev', 'reason': set([('fd', 'flags')])},
8879                  {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
8880                  {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
8881                  {'call': 'pread64', 'reason': set([('fd', 'flags')])},
8882                  {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
8883                  {'call': 'read', 'reason': set([('fd', 'flags')])},
8884                  {'call': 'fchown', 'reason': set([('fd', 'flags')])},
8885                  {'call': 'mq_timedreceive',
8886                   'reason': set([('fd', 'flags')])},
8887                  {'call': 'utime', 'reason': set([('fd', 'flags')])},
8888                  {'call': 'fsync', 'reason': set([('fd', 'flags')])},
8889                  {'call': 'bpf', 'reason': set([('fd', 'flags')])},
8890                  {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
8891                  {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
8892                  {'call': 'sendto', 'reason': set([('fd', 'flags')])},
8893                  {'call': 'tee', 'reason': set([('fd', 'flags')])},
8894                  {'call': 'sync_file_range',
8895                   'reason': set([('fd', 'flags')])},
8896                  {'call': 'lseek', 'reason': set([('fd', 'flags')])},
8897                  {'call': 'connect', 'reason': set([('fd', 'flags')])},
8898                  {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
8899                  {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
8900                  {'call': 'flock', 'reason': set([('fd', 'flags')])},
8901                  {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
8902                  {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
8903                  {'call': 'accept4', 'reason': set([('fd', 'flags')])},
```



```
8904                  {'call': 'inotify_rm_watch',
8905                   'reason': set([('fd', 'flags')])},
8906                  {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
8907                  {'call': 'inotify_add_watch',
8908                   'reason': set([('fd', 'flags')])},
8909                  {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
8910                  {'call': 'splice', 'reason': set([('fd', 'flags')])},
8911                  {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
8912                  {'call': 'preadv', 'reason': set([('fd', 'flags')])},
8913                  {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
8914                  {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
8915                  {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
8916                  {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
8917                  {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
8918                  {'call': 'perf_event_open',
8919                   'reason': set([('fd', 'flags')])},
8920                  {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
8921                  {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
8922                  {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
8923                  {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
8924                  {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
8925                  {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
8926                  {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
8927                  {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
8928                  {'call': 'listen', 'reason': set([('fd', 'flags')])},
8929                  {'call': 'copy_file_range',
8930                   'reason': set([('fd', 'flags')])},
8931                  {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
8932                  {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
8933                  {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
8934                  {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
8935                  {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
8936                  {'call': 'llseek', 'reason': set([('fd', 'flags')])},
8937                  {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
8938                  {'call': 'readv', 'reason': set([('fd', 'flags')])},
8939                  {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
8940                  {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
8941                  {'call': 'write', 'reason': set([('fd', 'flags')])},
8942                  {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
8943                  {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
8944                  {'call': 'bind', 'reason': set([('fd', 'flags')])},
8945                  {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
8946                  {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
8947  'olduname': [{'call': 'keyctl',
8948                'reason': set([('mm_segment_t', 'seg'),
8949                               ('task_struct', 'personality')])},
8950               {'call': 'rt_sigtimedwait',
8951                'reason': set([('mm_segment_t', 'seg'),
8952                               ('task_struct', 'personality')])},
8953               {'call': 'msgrcv',
8954                'reason': set([('mm_segment_t', 'seg'),
8955                               ('task_struct', 'personality')])},
8956               {'call': 'kill',
8957                'reason': set([('mm_segment_t', 'seg'),
8958                               ('task_struct', 'personality')])},
8959               {'call': 'sched_getaffinity',
8960                'reason': set([('mm_segment_t', 'seg'),
8961                               ('task_struct', 'personality')])},
8962               {'call': 'sched_setparam',
8963                'reason': set([('mm_segment_t', 'seg'),
8964                               ('task_struct', 'personality')])},
8965               {'call': 'ioprio_set',
8966                'reason': set([('mm_segment_t', 'seg'),
8967                               ('task_struct', 'personality')])},
8968               {'call': 'personality',
8969                'reason': set([('task_struct', 'personality')])},
```

```
8970                {'call': 'getppid',
8971                 'reason': set([('mm_segment_t', 'seg'),
8972                                ('task_struct', 'personality')])},
8973                {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
8974                {'call': 'mq_timedreceive',
8975                 'reason': set([('mm_segment_t', 'seg'),
8976                                ('task_struct', 'personality')])},
8977                {'call': 'capget',
8978                 'reason': set([('mm_segment_t', 'seg'),
8979                                ('task_struct', 'personality')])},
8980                {'call': 'sched_setaffinity',
8981                 'reason': set([('mm_segment_t', 'seg'),
8982                                ('task_struct', 'personality')])},
8983                {'call': 'signal',
8984                 'reason': set([('mm_segment_t', 'seg'),
8985                                ('task_struct', 'personality')])},
8986                {'call': 'semtimedop',
8987                 'reason': set([('mm_segment_t', 'seg'),
8988                                ('task_struct', 'personality')])},
8989                {'call': 'umount',
8990                 'reason': set([('mm_segment_t', 'seg'),
8991                                ('task_struct', 'personality')])},
8992                {'call': 'sched_rr_get_interval',
8993                 'reason': set([('mm_segment_t', 'seg'),
8994                                ('task_struct', 'personality')])},
8995                {'call': 'rt_sigprocmask',
8996                 'reason': set([('mm_segment_t', 'seg'),
8997                                ('task_struct', 'personality')])},
8998                {'call': 'setsid',
8999                 'reason': set([('mm_segment_t', 'seg'),
9000                                ('task_struct', 'personality')])},
9001                {'call': 'sigaltstack',
9002                 'reason': set([('mm_segment_t', 'seg'),
9003                                ('task_struct', 'personality')])},
9004                {'call': 'sched_setattr',
9005                 'reason': set([('mm_segment_t', 'seg'),
9006                                ('task_struct', 'personality')])},
9007                {'call': 'migrate_pages',
9008                 'reason': set([('mm_segment_t', 'seg'),
9009                                ('task_struct', 'personality')])},
9010                {'call': 'getitimer',
9011                 'reason': set([('mm_segment_t', 'seg'),
9012                                ('task_struct', 'personality')])},
9013                {'call': 'setpgid',
9014                 'reason': set([('mm_segment_t', 'seg'),
9015                                ('task_struct', 'personality')])},
9016                {'call': 'getsid',
9017                 'reason': set([('mm_segment_t', 'seg'),
9018                                ('task_struct', 'personality')])},
9019                {'call': 'prlimit64',
9020                 'reason': set([('mm_segment_t', 'seg'),
9021                                ('task_struct', 'personality')])},
9022                {'call': 'perf_event_open',
9023                 'reason': set([('mm_segment_t', 'seg'),
9024                                ('task_struct', 'personality')])},
9025                {'call': 'rt_sigaction',
9026                 'reason': set([('mm_segment_t', 'seg'),
9027                                ('task_struct', 'personality')])},
9028                {'call': 'getpgid',
9029                 'reason': set([('mm_segment_t', 'seg'),
9030                                ('task_struct', 'personality')])},
9031                {'call': 'getpriority',
9032                 'reason': set([('mm_segment_t', 'seg'),
9033                                ('task_struct', 'personality')])},
9034                {'call': 'sigaction',
9035                 'reason': set([('mm_segment_t', 'seg'),
```



```
9036                                ('task_struct', 'personality')])},
9037                {'call': 'setns',
9038                 'reason': set([('mm_segment_t', 'seg'),
9039                                ('task_struct', 'personality')])},
9040                {'call': 'fork',
9041                 'reason': set([('mm_segment_t', 'seg'),
9042                                ('task_struct', 'personality')])},
9043                {'call': 'get_robust_list',
9044                 'reason': set([('mm_segment_t', 'seg'),
9045                                ('task_struct', 'personality')])},
9046                {'call': 'mq_timedsend',
9047                 'reason': set([('mm_segment_t', 'seg'),
9048                                ('task_struct', 'personality')])},
9049                {'call': 'sched_getscheduler',
9050                 'reason': set([('mm_segment_t', 'seg'),
9051                                ('task_struct', 'personality')])},
9052                {'call': 'ptrace',
9053                 'reason': set([('mm_segment_t', 'seg'),
9054                                ('task_struct', 'personality')])},
9055                {'call': 'sched_getattr',
9056                 'reason': set([('mm_segment_t', 'seg'),
9057                                ('task_struct', 'personality')])},
9058                {'call': 'getrusage',
9059                 'reason': set([('mm_segment_t', 'seg'),
9060                                ('task_struct', 'personality')])},
9061                {'call': 'sched_setscheduler',
9062                 'reason': set([('mm_segment_t', 'seg'),
9063                                ('task_struct', 'personality')])},
9064                {'call': 'setitimer',
9065                 'reason': set([('mm_segment_t', 'seg'),
9066                                ('task_struct', 'personality')])},
9067                {'call': 'ioprio_get',
9068                 'reason': set([('mm_segment_t', 'seg'),
9069                                ('task_struct', 'personality')])},
9070                {'call': 'vfork',
9071                 'reason': set([('mm_segment_t', 'seg'),
9072                                ('task_struct', 'personality')])},
9073                {'call': 'prctl',
9074                 'reason': set([('mm_segment_t', 'seg'),
9075                                ('task_struct', 'personality')])},
9076                {'call': 'move_pages',
9077                 'reason': set([('mm_segment_t', 'seg'),
9078                                ('task_struct', 'personality')])},
9079                {'call': 'setpriority',
9080                 'reason': set([('mm_segment_t', 'seg'),
9081                                ('task_struct', 'personality')])},
9082                {'call': 'clone',
9083                 'reason': set([('mm_segment_t', 'seg'),
9084                                ('task_struct', 'personality')])},
9085                {'call': 'sched_getparam',
9086                 'reason': set([('mm_segment_t', 'seg'),
9087                                ('task_struct', 'personality')])}],
9088  'open_by_handle_at': [{'call': 'eventfd2',
9089                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9090                        {'call': 'swapoff',
9091                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9092                        {'call': 'pivot_root',
9093                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9094                        {'call': 'memfd_create',
9095                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9096                        {'call': 'remap_file_pages',
9097                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9098                        {'call': 'dup3',
9099                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
9100                        {'call': 'unshare',
9101                         'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```

```
                      {'call': 'epoll_create1',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'epoll_ctl',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'flock',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'openat',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'lookup_dcookie',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'uselib',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'accept4',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'socketpair',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'getcwd',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'shmat',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'socket',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'pipe2',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'perf_event_open',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'shmdt',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'quotactl',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'acct',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'open',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'dup',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'setns',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'shmctl',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'swapon',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'mmap_pgoff',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                      {'call': 'mq_open',
                       'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'perf_event_open': [{'call': 'syncfs',
                      'reason': set([('fd', 'flags'),
                                     ('list_head', 'prev')])},
                     {'call': 'keyctl',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'rt_sigtimedwait',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'vmsplice',
                      'reason': set([('fd', 'flags'),
                                     ('list_head', 'prev')])},
                     {'call': 'msgrcv',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'eventfd2',
                      'reason': set([('file', 'f_op'),
```



```
                                     ('list_head', 'prev')])},
                     {'call': 'mq_unlink',
                      'reason': set([('list_head', 'prev')])},
                     {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                     {'call': 'kill',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'swapoff',
                      'reason': set([('file', 'f_op'),
                                     ('list_head', 'prev')])},
                     {'call': 'fremovexattr',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'readahead',
                      'reason': set([('fd', 'flags'),
                                     ('list_head', 'prev')])},
                     {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                     {'call': 'timer_delete',
                      'reason': set([('list_head', 'prev')])},
                     {'call': 'sched_getaffinity',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'writev', 'reason': set([('fd', 'flags')])},
                     {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                     {'call': 'sched_setparam',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'fchmod',
                      'reason': set([('fd', 'flags'),
                                     ('list_head', 'prev')])},
                     {'call': 'setgid',
                      'reason': set([('list_head', 'prev')])},
                     {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                     {'call': 'pivot_root',
                      'reason': set([('list_head', 'prev')])},
                     {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                     {'call': 'memfd_create',
                      'reason': set([('file', 'f_op'),
                                     ('list_head', 'prev')])},
                     {'call': 'ioprio_set',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'delete_module',
                      'reason': set([('list_head', 'prev')])},
                     {'call': 'remap_file_pages',
                      'reason': set([('file', 'f_op'),
                                     ('list_head', 'prev')])},
                     {'call': 'dup3',
                      'reason': set([('file', 'f_op'),
                                     ('list_head', 'prev')])},
                     {'call': 'readlinkat',
                      'reason': set([('list_head', 'prev')])},
                     {'call': 'read', 'reason': set([('fd', 'flags')])},
                     {'call': 'io_getevents',
                      'reason': set([('list_head', 'prev')])},
                     {'call': 'getppid',
                      'reason': set([('list_head', 'prev'),
                                     ('mm_segment_t', 'seg'),
                                     ('task_struct', 'flags')])},
                     {'call': 'fchown',
                      'reason': set([('fd', 'flags'),
                                     ('list_head', 'prev')])},
                     {'call': 'ioperm',
```

```
                     'reason': set([('mm_segment_t', 'seg')])},
                    {'call': 'mq_timedreceive',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'capget',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'utime', 'reason': set([('fd', 'flags')])},
                    {'call': 'sched_setaffinity',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'ustat',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                    {'call': 'bpf',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'unshare',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'signal',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'setreuid',
                     'reason': set([('list_head', 'prev'),
                                    ('task_struct', 'flags')])},
                    {'call': 'semtimedop',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'umount',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                    {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'timer_create',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                    {'call': 'mkdirat',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sched_rr_get_interval',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'epoll_create1',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'timerfd_gettime',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'tee',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'semctl',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sync_file_range',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                    {'call': 'connect', 'reason': set([('fd', 'flags')])},
                    {'call': 'getsockname',
                     'reason': set([('fd', 'flags')])},
```



```
                    {'call': 'epoll_ctl',
                     'reason': set([('fd', 'flags'),
                                    ('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'flock',
                     'reason': set([('fd', 'flags'),
                                    ('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                    {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                    {'call': 'openat',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'lookup_dcookie',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'uselib',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'renameat2',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'rt_sigprocmask',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'accept4',
                     'reason': set([('fd', 'flags'),
                                    ('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'msgctl',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'reboot',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'setsid',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'set_trip_temp',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sigaltstack',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'sched_setattr',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'old_readdir',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'inotify_rm_watch',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'socketpair',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                    {'call': 'migrate_pages',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'getitimer',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'fchmodat',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'setpgid',
```

```
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'init_module',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'setresgid',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getcwd',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('fd', 'flags'),
                                ('list_head', 'prev')])},
                {'call': 'get_trip_temp',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                {'call': 'timer_settime',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'setregid',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'splice',
                 'reason': set([('fd', 'flags'),
                                ('list_head', 'prev')])},
                {'call': 'ftruncate',
                 'reason': set([('fd', 'flags'),
                                ('list_head', 'prev')])},
                {'call': 'timer_gettime',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                {'call': 'getpeername',
                 'reason': set([('fd', 'flags')])},
                {'call': 'getsid',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'shmat',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev')])},
                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'mknodat',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'socket',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev')])},
                {'call': 'symlinkat',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'pipe2',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev')])},
                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                {'call': 'ioctl',
                 'reason': set([('fd', 'flags'),
                                ('list_head', 'prev')])},
                {'call': 'prlimit64',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                {'call': 'linkat',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getgroups16',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'shmdt',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev')])},
                {'call': 'pwritev64v2',
                 'reason': set([('fd', 'flags')])},
```



```
                {'call': 'quotactl',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'rt_sigaction',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                {'call': 'request_key',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getpgid',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'brk', 'reason': set([('list_head', 'prev')])},
                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                {'call': 'acct',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev')])},
                {'call': 'open',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev')])},
                {'call': 'unlink',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'exit_group',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'getpriority',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'sigaction',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'mq_getsetattr',
                 'reason': set([('fd', 'flags'),
                                ('list_head', 'prev')])},
                {'call': 'faccessat',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'rmdir',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'dup',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                {'call': 'setgroups16',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'setns',
                 'reason': set([('file', 'f_op'),
                                ('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                {'call': 'listen', 'reason': set([('fd', 'flags')])},
                {'call': 'fork',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
                                ('task_struct', 'flags')])},
                {'call': 'get_mempolicy',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'io_submit',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'get_robust_list',
                 'reason': set([('list_head', 'prev'),
                                ('mm_segment_t', 'seg'),
```

```
                                    ('task_struct', 'flags')])},
                    {'call': 'copy_file_range',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'sched_yield',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'ptrace',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'shmctl',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                    {'call': 'munlockall',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'swapon',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                    {'call': 'pkey_mprotect',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'madvise',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'epoll_wait',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'sched_getattr',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'fchownat',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'getrusage',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'timerfd_settime',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'setresuid',
                     'reason': set([('list_head', 'prev'),
                                    ('task_struct', 'flags')])},
                    {'call': 'setitimer',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'ioprio_get',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'vfork',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
```



```
                                    ('task_struct', 'flags')])},
                    {'call': 'setuid',
                     'reason': set([('list_head', 'prev'),
                                    ('task_struct', 'flags')])},
                    {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                    {'call': 'io_setup',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'mprotect',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'mmap_pgoff',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'mremap',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'io_destroy',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'mbind',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                    {'call': 'readv', 'reason': set([('fd', 'flags')])},
                    {'call': 'prctl',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'move_pages',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'timerfd_create',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                    {'call': 'modify_ldt',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'getgroups',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                    {'call': 'dup2', 'reason': set([('list_head', 'prev')])},
                    {'call': 'get_curr_temp',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'msgsnd',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'write', 'reason': set([('fd', 'flags')])},
                    {'call': 'munlock',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'setpriority',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'inotify_init1',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'mincore',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'mq_notify',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'sendfile',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'timer_getoverrun',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'kexec_load',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'clone',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
```

```
                    {'call': 'mq_open',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'setgroups',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'unlinkat',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sched_getparam',
                     'reason': set([('list_head', 'prev'),
                                    ('mm_segment_t', 'seg'),
                                    ('task_struct', 'flags')])},
                    {'call': 'io_cancel',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'open_by_handle_at',
                     'reason': set([('file', 'f_op'),
                                    ('list_head', 'prev')])},
                    {'call': 'bind', 'reason': set([('fd', 'flags')])},
                    {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'finit_module',
                     'reason': set([('list_head', 'prev')])},
                    {'call': 'sendfile64',
                     'reason': set([('fd', 'flags'),
                                    ('list_head', 'prev')])},
                    {'call': 'mlockall',
                     'reason': set([('list_head', 'prev')])}],
 'pivot_root': [{'call': 'eventfd2',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mq_unlink',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'swapoff',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'memfd_create',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'remap_file_pages',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'dup3',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'unshare',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'umount',
                 'reason': set([('mount', 'mnt_ns'),
                                ('mount', 'mnt_parent'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'mkdirat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'epoll_create1',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'epoll_ctl',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'flock',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'openat',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'lookup_dcookie',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'uselib',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'renameat2',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'accept4',
```



```
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'socketpair',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'getcwd',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent'),
                                ('mount', 'mnt_ns'),
                                ('mount', 'mnt_parent'),
                                ('path', 'dentry'),
                                ('path', 'mnt'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'ftruncate',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'shmat',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mknodat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'socket',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'symlinkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'pipe2',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'perf_event_open',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'linkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'shmdt',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'quotactl',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'acct',
                 'reason': set([('path', 'dentry'),
                                ('path', 'mnt'),
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'open',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'unlink',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'rmdir',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'dup',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'setns',
                 'reason': set([('nsproxy', 'mnt_ns'),
                                ('path', 'dentry'),
                                ('path', 'mnt')])},
                {'call': 'shmctl',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'swapon',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mmap_pgoff',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                {'call': 'mq_open',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent'),
                                ('path', 'dentry'),
                                ('path', 'mnt'),
```

```
                                ('vfsmount', 'mnt_flags'),
                                ('vfsmount', 'mnt_root')])},
                {'call': 'unlinkat',
                 'reason': set([('dentry', 'd_inode'),
                                ('dentry', 'd_parent')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'pkey_alloc': [{'call': 'swapoff',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'remap_file_pages',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_getevents',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'pkey_free',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'migrate_pages',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'shmdt',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'brk',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'get_mempolicy',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'getrusage',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_setup',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'mremap',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_destroy',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'mbind',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'prctl',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'move_pages',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'modify_ldt',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'mincore',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])},
                {'call': 'io_cancel',
                 'reason': set([('mm_context_t', 'pkey_allocation_map')])}],
 'pkey_mprotect': [{'call': 'keyctl',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'msgrcv',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'kill',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'sched_setparam',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'ioprio_set',
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                   {'call': 'personality',
                    'reason': set([('task_struct', 'personality')])},
```



```
                {'call': 'remap_file_pages',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'capget',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'signal',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'umount',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'shmdt',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'getpgid',
```

```
                    'reason': set([('task_struct', 'mm'),
                                   ('task_struct', 'personality')])},
                {'call': 'brk',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'get_mempolicy',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'munlockall',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'madvise',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'mprotect',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'mremap',
                 'reason': set([('vm_area_struct', 'vm_end'),
```



```
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality'),
                                ('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'munlock',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'mincore',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'mm'),
                                ('task_struct', 'personality')])},
                {'call': 'mlockall',
                 'reason': set([('vm_area_struct', 'vm_end'),
                                ('vm_area_struct', 'vm_flags'),
                                ('vm_area_struct', 'vm_start')])}],
 'poll': [{'call': 'ppoll', 'reason': set([('poll_list', 'len')])}],
 'ppoll': [{'call': 'keyctl',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('task_struct', 'personality'),
                           ('timespec', 'tv_nsec'),
                           ('timespec', 'tv_sec')])},
           {'call': 'msgrcv',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'mq_unlink',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
           {'call': 'swapoff',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'sched_getaffinity',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'sched_setparam',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'fchmod',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'memfd_create',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'ioprio_set',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'personality',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'readlinkat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'io_getevents',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'getppid',
            'reason': set([('task_struct', 'personality')])},
           {'call': 'fchown',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
```

```
10026           {'call': 'mq_timedreceive',
10027            'reason': set([('task_struct', 'personality'),
10028                           ('timespec', 'tv_nsec'),
10029                           ('timespec', 'tv_sec')])},
10030           {'call': 'capget',
10031            'reason': set([('task_struct', 'personality')])},
10032           {'call': 'utime',
10033            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10034           {'call': 'sched_setaffinity',
10035            'reason': set([('task_struct', 'personality')])},
10036           {'call': 'signal',
10037            'reason': set([('task_struct', 'personality')])},
10038           {'call': 'semtimedop',
10039            'reason': set([('task_struct', 'personality'),
10040                           ('timespec', 'tv_nsec'),
10041                           ('timespec', 'tv_sec')])},
10042           {'call': 'umount',
10043            'reason': set([('task_struct', 'personality')])},
10044           {'call': 'settimeofday',
10045            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10046           {'call': 'sched_rr_get_interval',
10047            'reason': set([('task_struct', 'personality'),
10048                           ('timespec', 'tv_nsec'),
10049                           ('timespec', 'tv_sec')])},
10050           {'call': 'timerfd_gettime',
10051            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10052           {'call': 'pselect6',
10053            'reason': set([('compat_timespec', 'tv_nsec'),
10054                           ('compat_timespec', 'tv_sec'),
10055                           ('timespec', 'tv_nsec'),
10056                           ('timespec', 'tv_sec')])},
10057           {'call': 'uselib',
10058            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10059           {'call': 'rt_sigprocmask',
10060            'reason': set([('task_struct', 'personality')])},
10061           {'call': 'setsid',
10062            'reason': set([('task_struct', 'personality')])},
10063           {'call': 'sigaltstack',
10064            'reason': set([('task_struct', 'personality')])},
10065           {'call': 'sched_setattr',
10066            'reason': set([('task_struct', 'personality')])},
10067           {'call': 'migrate_pages',
10068            'reason': set([('task_struct', 'personality')])},
10069           {'call': 'getitimer',
10070            'reason': set([('task_struct', 'personality')])},
10071           {'call': 'fchmodat',
10072            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10073           {'call': 'setpgid',
10074            'reason': set([('task_struct', 'personality')])},
10075           {'call': 'inotify_add_watch',
10076            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10077           {'call': 'timer_settime',
10078            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10079           {'call': 'ftruncate',
10080            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10081           {'call': 'timer_gettime',
10082            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10083           {'call': 'getsid',
10084            'reason': set([('task_struct', 'personality')])},
10085           {'call': 'ioctl',
10086            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10087           {'call': 'prlimit64',
10088            'reason': set([('task_struct', 'personality')])},
10089           {'call': 'perf_event_open',
10090            'reason': set([('task_struct', 'personality')])},
10091           {'call': 'linkat',
```



```
10092            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10093           {'call': 'stime',
10094            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10095           {'call': 'rt_sigaction',
10096            'reason': set([('task_struct', 'personality')])},
10097           {'call': 'futimesat',
10098            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10099           {'call': 'getpgid',
10100            'reason': set([('task_struct', 'personality')])},
10101           {'call': 'poll',
10102            'reason': set([('poll_list', 'len'),
10103                           ('timespec', 'tv_nsec'),
10104                           ('timespec', 'tv_sec')])},
10105           {'call': 'select',
10106            'reason': set([('compat_timespec', 'tv_nsec'),
10107                           ('compat_timespec', 'tv_sec'),
10108                           ('timespec', 'tv_nsec'),
10109                           ('timespec', 'tv_sec')])},
10110           {'call': 'unlink',
10111            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10112           {'call': 'getpriority',
10113            'reason': set([('task_struct', 'personality')])},
10114           {'call': 'sigaction',
10115            'reason': set([('task_struct', 'personality')])},
10116           {'call': 'nanosleep',
10117            'reason': set([('compat_timespec', 'tv_nsec'),
10118                           ('compat_timespec', 'tv_sec'),
10119                           ('timespec', 'tv_nsec'),
10120                           ('timespec', 'tv_sec')])},
10121           {'call': 'mq_getsetattr',
10122            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10123           {'call': 'faccessat',
10124            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10125           {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
10126           {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
10127           {'call': 'get_robust_list',
10128            'reason': set([('task_struct', 'personality')])},
10129           {'call': 'mq_timedsend',
10130            'reason': set([('task_struct', 'personality'),
10131                           ('timespec', 'tv_nsec'),
10132                           ('timespec', 'tv_sec')])},
10133           {'call': 'sched_getscheduler',
10134            'reason': set([('task_struct', 'personality')])},
10135           {'call': 'ptrace',
10136            'reason': set([('task_struct', 'personality')])},
10137           {'call': 'swapon',
10138            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10139           {'call': 'epoll_wait',
10140            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10141           {'call': 'sched_getattr',
10142            'reason': set([('task_struct', 'personality')])},
10143           {'call': 'fchownat',
10144            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10145           {'call': 'getrusage',
10146            'reason': set([('task_struct', 'personality')])},
10147           {'call': 'fstat',
10148            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10149           {'call': 'timerfd_settime',
10150            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10151           {'call': 'sched_setscheduler',
10152            'reason': set([('task_struct', 'personality')])},
10153           {'call': 'setitimer',
10154            'reason': set([('task_struct', 'personality')])},
10155           {'call': 'ioprio_get',
10156            'reason': set([('task_struct', 'personality')])},
10157           {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
```

```
10158           {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
10159           {'call': 'move_pages',
10160            'reason': set([('task_struct', 'personality')])},
10161           {'call': 'setpriority',
10162            'reason': set([('task_struct', 'personality')])},
10163           {'call': 'mq_notify',
10164            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10165           {'call': 'sendfile',
10166            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10167           {'call': 'newfstat',
10168            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10169           {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
10170           {'call': 'clock_nanosleep',
10171            'reason': set([('compat_timespec', 'tv_nsec'),
10172                           ('compat_timespec', 'tv_sec'),
10173                           ('timespec', 'tv_nsec'),
10174                           ('timespec', 'tv_sec')])},
10175           {'call': 'unlinkat',
10176            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10177           {'call': 'sched_getparam',
10178            'reason': set([('task_struct', 'personality')])},
10179           {'call': 'futex',
10180            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10181           {'call': 'recvmmsg',
10182            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
10183           {'call': 'sendfile64',
10184            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])}],
10185  'prctl': [{'call': 'keyctl',
10186            'reason': set([('task_struct', 'flags'),
10187                           ('task_struct', 'personality'),
10188                           ('task_struct', 'timer_slack_ns')])},
10189           {'call': 'rt_sigtimedwait',
10190            'reason': set([('task_struct', 'flags'),
10191                           ('task_struct', 'personality'),
10192                           ('task_struct', 'timer_slack_ns')])},
10193           {'call': 'msgrcv',
10194            'reason': set([('task_struct', 'flags'),
10195                           ('task_struct', 'personality'),
10196                           ('task_struct', 'timer_slack_ns')])},
10197           {'call': 'kill',
10198            'reason': set([('task_struct', 'flags'),
10199                           ('task_struct', 'personality'),
10200                           ('task_struct', 'timer_slack_ns')])},
10201           {'call': 'swapoff', 'reason': set([('mm_struct', 'flags')])},
10202           {'call': 'sched_getaffinity',
10203            'reason': set([('task_struct', 'flags'),
10204                           ('task_struct', 'personality'),
10205                           ('task_struct', 'timer_slack_ns')])},
10206           {'call': 'sched_setparam',
10207            'reason': set([('task_struct', 'flags'),
10208                           ('task_struct', 'personality'),
10209                           ('task_struct', 'timer_slack_ns')])},
10210           {'call': 'ioprio_set',
10211            'reason': set([('task_struct', 'flags'),
10212                           ('task_struct', 'personality'),
10213                           ('task_struct', 'timer_slack_ns')])},
10214           {'call': 'personality',
10215            'reason': set([('task_struct', 'personality')])},
10216           {'call': 'remap_file_pages',
10217            'reason': set([('mm_struct', 'flags')])},
10218           {'call': 'io_getevents', 'reason': set([('mm_struct', 'flags')])},
10219           {'call': 'getppid',
10220            'reason': set([('task_struct', 'flags'),
10221                           ('task_struct', 'personality'),
10222                           ('task_struct', 'timer_slack_ns')])},
10223           {'call': 'mq_timedreceive',
```



```
10224            'reason': set([('task_struct', 'flags'),
10225                           ('task_struct', 'personality'),
10226                           ('task_struct', 'timer_slack_ns')])},
10227           {'call': 'capget',
10228            'reason': set([('task_struct', 'flags'),
10229                           ('task_struct', 'personality'),
10230                           ('task_struct', 'timer_slack_ns')])},
10231           {'call': 'sched_setaffinity',
10232            'reason': set([('task_struct', 'flags'),
10233                           ('task_struct', 'personality'),
10234                           ('task_struct', 'timer_slack_ns')])},
10235           {'call': 'signal',
10236            'reason': set([('task_struct', 'flags'),
10237                           ('task_struct', 'personality'),
10238                           ('task_struct', 'timer_slack_ns')])},
10239           {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
10240           {'call': 'semtimedop',
10241            'reason': set([('task_struct', 'flags'),
10242                           ('task_struct', 'personality'),
10243                           ('task_struct', 'timer_slack_ns')])},
10244           {'call': 'umount',
10245            'reason': set([('task_struct', 'flags'),
10246                           ('task_struct', 'personality'),
10247                           ('task_struct', 'timer_slack_ns')])},
10248           {'call': 'sched_rr_get_interval',
10249            'reason': set([('task_struct', 'flags'),
10250                           ('task_struct', 'personality'),
10251                           ('task_struct', 'timer_slack_ns')])},
10252           {'call': 'rt_sigprocmask',
10253            'reason': set([('task_struct', 'flags'),
10254                           ('task_struct', 'personality'),
10255                           ('task_struct', 'timer_slack_ns')])},
10256           {'call': 'setsid',
10257            'reason': set([('task_struct', 'flags'),
10258                           ('task_struct', 'personality'),
10259                           ('task_struct', 'timer_slack_ns')])},
10260           {'call': 'sigaltstack',
10261            'reason': set([('task_struct', 'flags'),
10262                           ('task_struct', 'personality'),
10263                           ('task_struct', 'timer_slack_ns')])},
10264           {'call': 'sched_setattr',
10265            'reason': set([('task_struct', 'flags'),
10266                           ('task_struct', 'personality'),
10267                           ('task_struct', 'timer_slack_ns')])},
10268           {'call': 'migrate_pages',
10269            'reason': set([('mm_struct', 'flags'),
10270                           ('task_struct', 'flags'),
10271                           ('task_struct', 'personality'),
10272                           ('task_struct', 'timer_slack_ns')])},
10273           {'call': 'getitimer',
10274            'reason': set([('task_struct', 'flags'),
10275                           ('task_struct', 'personality'),
10276                           ('task_struct', 'timer_slack_ns')])},
10277           {'call': 'setpgid',
10278            'reason': set([('task_struct', 'flags'),
10279                           ('task_struct', 'personality'),
10280                           ('task_struct', 'timer_slack_ns')])},
10281           {'call': 'getsid',
10282            'reason': set([('task_struct', 'flags'),
10283                           ('task_struct', 'personality'),
10284                           ('task_struct', 'timer_slack_ns')])},
10285           {'call': 'prlimit64',
10286            'reason': set([('task_struct', 'flags'),
10287                           ('task_struct', 'personality'),
10288                           ('task_struct', 'timer_slack_ns')])},
10289           {'call': 'perf_event_open',
```

```
10290             'reason': set([('task_struct', 'flags'),
10291                            ('task_struct', 'personality'),
10292                            ('task_struct', 'timer_slack_ns')])},
10293           {'call': 'shmdt', 'reason': set([('mm_struct', 'flags')])},
10294           {'call': 'rt_sigaction',
10295            'reason': set([('task_struct', 'flags'),
10296                           ('task_struct', 'personality'),
10297                           ('task_struct', 'timer_slack_ns')])},
10298           {'call': 'getpgid',
10299            'reason': set([('task_struct', 'flags'),
10300                           ('task_struct', 'personality'),
10301                           ('task_struct', 'timer_slack_ns')])},
10302           {'call': 'brk', 'reason': set([('mm_struct', 'flags')])},
10303           {'call': 'getpriority',
10304            'reason': set([('task_struct', 'flags'),
10305                           ('task_struct', 'personality'),
10306                           ('task_struct', 'timer_slack_ns')])},
10307           {'call': 'sigaction',
10308            'reason': set([('task_struct', 'flags'),
10309                           ('task_struct', 'personality'),
10310                           ('task_struct', 'timer_slack_ns')])},
10311           {'call': 'setns',
10312            'reason': set([('task_struct', 'flags'),
10313                           ('task_struct', 'personality'),
10314                           ('task_struct', 'timer_slack_ns')])},
10315           {'call': 'fork',
10316            'reason': set([('task_struct', 'flags'),
10317                           ('task_struct', 'personality'),
10318                           ('task_struct', 'timer_slack_ns')])},
10319           {'call': 'get_mempolicy', 'reason': set([('mm_struct', 'flags')])},
10320           {'call': 'get_robust_list',
10321            'reason': set([('task_struct', 'flags'),
10322                           ('task_struct', 'personality'),
10323                           ('task_struct', 'timer_slack_ns')])},
10324           {'call': 'mq_timedsend',
10325            'reason': set([('task_struct', 'flags'),
10326                           ('task_struct', 'personality'),
10327                           ('task_struct', 'timer_slack_ns')])},
10328           {'call': 'sched_getscheduler',
10329            'reason': set([('task_struct', 'flags'),
10330                           ('task_struct', 'personality'),
10331                           ('task_struct', 'timer_slack_ns')])},
10332           {'call': 'ptrace',
10333            'reason': set([('task_struct', 'flags'),
10334                           ('task_struct', 'personality'),
10335                           ('task_struct', 'timer_slack_ns')])},
10336           {'call': 'sched_getattr',
10337            'reason': set([('task_struct', 'flags'),
10338                           ('task_struct', 'personality'),
10339                           ('task_struct', 'timer_slack_ns')])},
10340           {'call': 'getrusage',
10341            'reason': set([('mm_struct', 'flags'),
10342                           ('task_struct', 'flags'),
10343                           ('task_struct', 'personality'),
10344                           ('task_struct', 'timer_slack_ns')])},
10345           {'call': 'sched_setscheduler',
10346            'reason': set([('task_struct', 'flags'),
10347                           ('task_struct', 'personality'),
10348                           ('task_struct', 'timer_slack_ns')])},
10349           {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
10350           {'call': 'setitimer',
10351            'reason': set([('task_struct', 'flags'),
10352                           ('task_struct', 'personality'),
10353                           ('task_struct', 'timer_slack_ns')])},
10354           {'call': 'ioprio_get',
10355            'reason': set([('task_struct', 'flags'),
```



```
10356                           ('task_struct', 'personality'),
10357                           ('task_struct', 'timer_slack_ns')])},
10358           {'call': 'vfork',
10359            'reason': set([('task_struct', 'flags'),
10360                           ('task_struct', 'personality'),
10361                           ('task_struct', 'timer_slack_ns')])},
10362           {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
10363           {'call': 'io_setup', 'reason': set([('mm_struct', 'flags')])},
10364           {'call': 'mremap', 'reason': set([('mm_struct', 'flags')])},
10365           {'call': 'io_destroy', 'reason': set([('mm_struct', 'flags')])},
10366           {'call': 'mbind', 'reason': set([('mm_struct', 'flags')])},
10367           {'call': 'move_pages',
10368            'reason': set([('mm_struct', 'flags'),
10369                           ('task_struct', 'flags'),
10370                           ('task_struct', 'personality'),
10371                           ('task_struct', 'timer_slack_ns')])},
10372           {'call': 'modify_ldt', 'reason': set([('mm_struct', 'flags')])},
10373           {'call': 'setpriority',
10374            'reason': set([('task_struct', 'flags'),
10375                           ('task_struct', 'personality'),
10376                           ('task_struct', 'timer_slack_ns')])},
10377           {'call': 'mincore', 'reason': set([('mm_struct', 'flags')])},
10378           {'call': 'clone',
10379            'reason': set([('task_struct', 'flags'),
10380                           ('task_struct', 'personality'),
10381                           ('task_struct', 'timer_slack_ns')])},
10382           {'call': 'sched_getparam',
10383            'reason': set([('task_struct', 'flags'),
10384                           ('task_struct', 'personality'),
10385                           ('task_struct', 'timer_slack_ns')])},
10386           {'call': 'io_cancel', 'reason': set([('mm_struct', 'flags')])}],
10387  'pread64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
10388              {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
10389              {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
10390              {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
10391              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
10392              {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
10393              {'call': 'readahead', 'reason': set([('fd', 'flags')])},
10394              {'call': 'getdents', 'reason': set([('fd', 'flags')])},
10395              {'call': 'writev', 'reason': set([('fd', 'flags')])},
10396              {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
10397              {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
10398              {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
10399              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
10400              {'call': 'remap_file_pages',
10401               'reason': set([('file', 'f_mode')])},
10402              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
10403              {'call': 'read', 'reason': set([('fd', 'flags')])},
10404              {'call': 'fchown', 'reason': set([('fd', 'flags')])},
10405              {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
10406              {'call': 'utime', 'reason': set([('fd', 'flags')])},
10407              {'call': 'fsync', 'reason': set([('fd', 'flags')])},
10408              {'call': 'bpf', 'reason': set([('fd', 'flags')])},
10409              {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
10410              {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
10411              {'call': 'sendto', 'reason': set([('fd', 'flags')])},
10412              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
10413              {'call': 'tee', 'reason': set([('fd', 'flags')])},
10414              {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
10415              {'call': 'lseek', 'reason': set([('fd', 'flags')])},
10416              {'call': 'connect', 'reason': set([('fd', 'flags')])},
10417              {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
10418              {'call': 'epoll_ctl',
10419               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10420              {'call': 'flock',
10421               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
```

```
10422             {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
10423             {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
10424             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
10425             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
10426             {'call': 'accept4',
10427              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10428             {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
10429             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
10430             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
10431             {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
10432             {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
10433             {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
10434             {'call': 'splice', 'reason': set([('fd', 'flags')])},
10435             {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
10436             {'call': 'preadv', 'reason': set([('fd', 'flags')])},
10437             {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
10438             {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
10439             {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
10440             {'call': 'socket', 'reason': set([('file', 'f_mode')])},
10441             {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
10442             {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
10443             {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
10444             {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
10445             {'call': 'perf_event_open',
10446              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10447             {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
10448             {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
10449             {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
10450             {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
10451             {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
10452             {'call': 'acct', 'reason': set([('file', 'f_mode')])},
10453             {'call': 'open', 'reason': set([('file', 'f_mode')])},
10454             {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
10455             {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
10456             {'call': 'dup', 'reason': set([('file', 'f_mode')])},
10457             {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
10458             {'call': 'setns', 'reason': set([('file', 'f_mode')])},
10459             {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
10460             {'call': 'listen', 'reason': set([('fd', 'flags')])},
10461             {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
10462             {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
10463             {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
10464             {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
10465             {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
10466             {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
10467             {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
10468             {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
10469             {'call': 'llseek', 'reason': set([('fd', 'flags')])},
10470             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
10471             {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
10472             {'call': 'readv', 'reason': set([('fd', 'flags')])},
10473             {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
10474             {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
10475             {'call': 'write', 'reason': set([('fd', 'flags')])},
10476             {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
10477             {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
10478             {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
10479             {'call': 'open_by_handle_at',
10480              'reason': set([('file', 'f_mode')])},
10481             {'call': 'bind', 'reason': set([('fd', 'flags')])},
10482             {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
10483             {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
10484  'preadv': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
10485             {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
10486             {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
10487             {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
```



```
10488             {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
10489             {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
10490             {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
10491             {'call': 'flock', 'reason': set([('file', 'f_mode')])},
10492             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
10493             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
10494             {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
10495             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
10496             {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
10497             {'call': 'socket', 'reason': set([('file', 'f_mode')])},
10498             {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
10499             {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
10500             {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
10501             {'call': 'acct', 'reason': set([('file', 'f_mode')])},
10502             {'call': 'open', 'reason': set([('file', 'f_mode')])},
10503             {'call': 'dup', 'reason': set([('file', 'f_mode')])},
10504             {'call': 'setns', 'reason': set([('file', 'f_mode')])},
10505             {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
10506             {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
10507             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
10508             {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
10509             {'call': 'open_by_handle_at',
10510              'reason': set([('file', 'f_mode')])}],
10511  'preadv2': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
10512              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
10513              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
10514              {'call': 'remap_file_pages',
10515               'reason': set([('file', 'f_mode')])},
10516              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
10517              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
10518              {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
10519              {'call': 'flock', 'reason': set([('file', 'f_mode')])},
10520              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
10521              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
10522              {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
10523              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
10524              {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
10525              {'call': 'socket', 'reason': set([('file', 'f_mode')])},
10526              {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
10527              {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
10528              {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
10529              {'call': 'acct', 'reason': set([('file', 'f_mode')])},
10530              {'call': 'open', 'reason': set([('file', 'f_mode')])},
10531              {'call': 'dup', 'reason': set([('file', 'f_mode')])},
10532              {'call': 'setns', 'reason': set([('file', 'f_mode')])},
10533              {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
10534              {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
10535              {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
10536              {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
10537              {'call': 'open_by_handle_at',
10538               'reason': set([('file', 'f_mode')])}],
10539  'preadv64': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
10540               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
10541               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
10542               {'call': 'remap_file_pages',
10543                'reason': set([('file', 'f_mode')])},
10544               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
10545               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
10546               {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
10547               {'call': 'flock', 'reason': set([('file', 'f_mode')])},
10548               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
10549               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
10550               {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
10551               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
10552               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
10553               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
```

```
10554               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
10555               {'call': 'perf_event_open',
10556                'reason': set([('file', 'f_mode')])},
10557               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
10558               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
10559               {'call': 'open', 'reason': set([('file', 'f_mode')])},
10560               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
10561               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
10562               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
10563               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
10564               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
10565               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
10566               {'call': 'open_by_handle_at',
10567                'reason': set([('file', 'f_mode')])}],
10568  'preadv64v2': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
10569                 {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
10570                 {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
10571                 {'call': 'remap_file_pages',
10572                  'reason': set([('file', 'f_mode')])},
10573                 {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
10574                 {'call': 'epoll_create1',
10575                  'reason': set([('file', 'f_mode')])},
10576                 {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
10577                 {'call': 'flock', 'reason': set([('file', 'f_mode')])},
10578                 {'call': 'openat', 'reason': set([('file', 'f_mode')])},
10579                 {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
10580                 {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
10581                 {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
10582                 {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
10583                 {'call': 'socket', 'reason': set([('file', 'f_mode')])},
10584                 {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
10585                 {'call': 'perf_event_open',
10586                  'reason': set([('file', 'f_mode')])},
10587                 {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
10588                 {'call': 'acct', 'reason': set([('file', 'f_mode')])},
10589                 {'call': 'open', 'reason': set([('file', 'f_mode')])},
10590                 {'call': 'dup', 'reason': set([('file', 'f_mode')])},
10591                 {'call': 'setns', 'reason': set([('file', 'f_mode')])},
10592                 {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
10593                 {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
10594                 {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
10595                 {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
10596                 {'call': 'open_by_handle_at',
10597                  'reason': set([('file', 'f_mode')])}],
10598  'prlimit64': [{'call': 'keyctl',
10599                 'reason': set([('cred', 'user_ns'),
10600                                ('task_struct', 'cred'),
10601                                ('task_struct', 'group_leader'),
10602                                ('task_struct', 'real_cred'),
10603                                ('task_struct', 'sighand')])},
10604                {'call': 'rt_sigtimedwait',
10605                 'reason': set([('task_struct', 'cred'),
10606                                ('task_struct', 'group_leader'),
10607                                ('task_struct', 'real_cred'),
10608                                ('task_struct', 'sighand')])},
10609                {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
10610                {'call': 'msgrcv',
10611                 'reason': set([('task_struct', 'cred'),
10612                                ('task_struct', 'group_leader'),
10613                                ('task_struct', 'real_cred'),
10614                                ('task_struct', 'sighand')])},
10615                {'call': 'kill',
10616                 'reason': set([('task_struct', 'cred'),
10617                                ('task_struct', 'group_leader'),
10618                                ('task_struct', 'real_cred'),
10619                                ('task_struct', 'sighand')])},
```



```
10620               {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
10621               {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
10622               {'call': 'sched_getaffinity',
10623                'reason': set([('task_struct', 'cred'),
10624                               ('task_struct', 'group_leader'),
10625                               ('task_struct', 'real_cred'),
10626                               ('task_struct', 'sighand')])},
10627               {'call': 'sched_setparam',
10628                'reason': set([('task_struct', 'cred'),
10629                               ('task_struct', 'group_leader'),
10630                               ('task_struct', 'real_cred'),
10631                               ('task_struct', 'sighand')])},
10632               {'call': 'setgid',
10633                'reason': set([('cred', 'egid'),
10634                               ('cred', 'gid'),
10635                               ('cred', 'sgid'),
10636                               ('cred', 'user_ns')])},
10637               {'call': 'ioprio_set',
10638                'reason': set([('cred', 'user_ns'),
10639                               ('task_struct', 'cred'),
10640                               ('task_struct', 'group_leader'),
10641                               ('task_struct', 'real_cred'),
10642                               ('task_struct', 'sighand')])},
10643               {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
10644               {'call': 'getppid',
10645                'reason': set([('task_struct', 'cred'),
10646                               ('task_struct', 'group_leader'),
10647                               ('task_struct', 'real_cred'),
10648                               ('task_struct', 'sighand')])},
10649               {'call': 'mq_timedreceive',
10650                'reason': set([('task_struct', 'cred'),
10651                               ('task_struct', 'group_leader'),
10652                               ('task_struct', 'real_cred'),
10653                               ('task_struct', 'sighand')])},
10654               {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
10655               {'call': 'capget',
10656                'reason': set([('task_struct', 'cred'),
10657                               ('task_struct', 'group_leader'),
10658                               ('task_struct', 'real_cred'),
10659                               ('task_struct', 'sighand')])},
10660               {'call': 'sched_setaffinity',
10661                'reason': set([('cred', 'user_ns'),
10662                               ('task_struct', 'cred'),
10663                               ('task_struct', 'group_leader'),
10664                               ('task_struct', 'real_cred'),
10665                               ('task_struct', 'sighand')])},
10666               {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
10667               {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
10668               {'call': 'signal',
10669                'reason': set([('task_struct', 'cred'),
10670                               ('task_struct', 'group_leader'),
10671                               ('task_struct', 'real_cred'),
10672                               ('task_struct', 'sighand')])},
10673               {'call': 'setreuid',
10674                'reason': set([('cred', 'euid'),
10675                               ('cred', 'suid'),
10676                               ('cred', 'uid'),
10677                               ('cred', 'user_ns')])},
10678               {'call': 'semtimedop',
10679                'reason': set([('task_struct', 'cred'),
10680                               ('task_struct', 'group_leader'),
10681                               ('task_struct', 'real_cred'),
10682                               ('task_struct', 'sighand')])},
10683               {'call': 'umount',
10684                'reason': set([('task_struct', 'cred'),
10685                               ('task_struct', 'group_leader'),
```

```
10686                                 ('task_struct', 'real_cred'),
10687                                 ('task_struct', 'sighand')])},
10688             {'call': 'sched_rr_get_interval',
10689              'reason': set([('task_struct', 'cred'),
10690                             ('task_struct', 'group_leader'),
10691                             ('task_struct', 'real_cred'),
10692                             ('task_struct', 'sighand')])},
10693             {'call': 'epoll_create1',
10694              'reason': set([('cred', 'user_ns')])},
10695             {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
10696             {'call': 'rt_sigprocmask',
10697              'reason': set([('task_struct', 'cred'),
10698                             ('task_struct', 'group_leader'),
10699                             ('task_struct', 'real_cred'),
10700                             ('task_struct', 'sighand')])},
10701             {'call': 'setsid',
10702              'reason': set([('task_struct', 'cred'),
10703                             ('task_struct', 'group_leader'),
10704                             ('task_struct', 'real_cred'),
10705                             ('task_struct', 'sighand')])},
10706             {'call': 'sigaltstack',
10707              'reason': set([('task_struct', 'cred'),
10708                             ('task_struct', 'group_leader'),
10709                             ('task_struct', 'real_cred'),
10710                             ('task_struct', 'sighand')])},
10711             {'call': 'sched_setattr',
10712              'reason': set([('task_struct', 'cred'),
10713                             ('task_struct', 'group_leader'),
10714                             ('task_struct', 'real_cred'),
10715                             ('task_struct', 'sighand')])},
10716             {'call': 'setrlimit',
10717              'reason': set([('rlimit', 'rlim_cur'),
10718                             ('rlimit', 'rlim_max')])},
10719             {'call': 'migrate_pages',
10720              'reason': set([('cred', 'user_ns'),
10721                             ('task_struct', 'cred'),
10722                             ('task_struct', 'group_leader'),
10723                             ('task_struct', 'real_cred'),
10724                             ('task_struct', 'sighand')])},
10725             {'call': 'getitimer',
10726              'reason': set([('task_struct', 'cred'),
10727                             ('task_struct', 'group_leader'),
10728                             ('task_struct', 'real_cred'),
10729                             ('task_struct', 'sighand')])},
10730             {'call': 'setpgid',
10731              'reason': set([('task_struct', 'cred'),
10732                             ('task_struct', 'group_leader'),
10733                             ('task_struct', 'real_cred'),
10734                             ('task_struct', 'sighand')])},
10735             {'call': 'setresgid',
10736              'reason': set([('cred', 'egid'),
10737                             ('cred', 'gid'),
10738                             ('cred', 'sgid'),
10739                             ('cred', 'user_ns')])},
10740             {'call': 'setregid',
10741              'reason': set([('cred', 'egid'),
10742                             ('cred', 'gid'),
10743                             ('cred', 'sgid'),
10744                             ('cred', 'user_ns')])},
10745             {'call': 'getsid',
10746              'reason': set([('task_struct', 'cred'),
10747                             ('task_struct', 'group_leader'),
10748                             ('task_struct', 'real_cred'),
10749                             ('task_struct', 'sighand')])},
10750             {'call': 'old_getrlimit',
10751              'reason': set([('rlimit', 'rlim_cur'),
```



```
10752                                 ('rlimit', 'rlim_max')])},
10753             {'call': 'perf_event_open',
10754              'reason': set([('task_struct', 'cred'),
10755                             ('task_struct', 'group_leader'),
10756                             ('task_struct', 'real_cred'),
10757                             ('task_struct', 'sighand')])},
10758             {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
10759             {'call': 'rt_sigaction',
10760              'reason': set([('task_struct', 'cred'),
10761                             ('task_struct', 'group_leader'),
10762                             ('task_struct', 'real_cred'),
10763                             ('task_struct', 'sighand')])},
10764             {'call': 'getpgid',
10765              'reason': set([('task_struct', 'cred'),
10766                             ('task_struct', 'group_leader'),
10767                             ('task_struct', 'real_cred'),
10768                             ('task_struct', 'sighand')])},
10769             {'call': 'getpriority',
10770              'reason': set([('cred', 'user_ns'),
10771                             ('task_struct', 'cred'),
10772                             ('task_struct', 'group_leader'),
10773                             ('task_struct', 'real_cred'),
10774                             ('task_struct', 'sighand')])},
10775             {'call': 'sigaction',
10776              'reason': set([('task_struct', 'cred'),
10777                             ('task_struct', 'group_leader'),
10778                             ('task_struct', 'real_cred'),
10779                             ('task_struct', 'sighand')])},
10780             {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
10781             {'call': 'setns',
10782              'reason': set([('task_struct', 'cred'),
10783                             ('task_struct', 'group_leader'),
10784                             ('task_struct', 'real_cred'),
10785                             ('task_struct', 'sighand')])},
10786             {'call': 'fork',
10787              'reason': set([('task_struct', 'cred'),
10788                             ('task_struct', 'group_leader'),
10789                             ('task_struct', 'real_cred'),
10790                             ('task_struct', 'sighand')])},
10791             {'call': 'get_robust_list',
10792              'reason': set([('task_struct', 'cred'),
10793                             ('task_struct', 'group_leader'),
10794                             ('task_struct', 'real_cred'),
10795                             ('task_struct', 'sighand')])},
10796             {'call': 'mq_timedsend',
10797              'reason': set([('task_struct', 'cred'),
10798                             ('task_struct', 'group_leader'),
10799                             ('task_struct', 'real_cred'),
10800                             ('task_struct', 'sighand')])},
10801             {'call': 'sched_getscheduler',
10802              'reason': set([('task_struct', 'cred'),
10803                             ('task_struct', 'group_leader'),
10804                             ('task_struct', 'real_cred'),
10805                             ('task_struct', 'sighand')])},
10806             {'call': 'ptrace',
10807              'reason': set([('task_struct', 'cred'),
10808                             ('task_struct', 'group_leader'),
10809                             ('task_struct', 'real_cred'),
10810                             ('task_struct', 'sighand')])},
10811             {'call': 'sched_getattr',
10812              'reason': set([('task_struct', 'cred'),
10813                             ('task_struct', 'group_leader'),
10814                             ('task_struct', 'real_cred'),
10815                             ('task_struct', 'sighand')])},
10816             {'call': 'getrusage',
10817              'reason': set([('task_struct', 'cred'),
```

```
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'sched_setscheduler',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'setresuid',
               'reason': set([('cred', 'euid'),
                                ('cred', 'suid'),
                                ('cred', 'uid'),
                                ('cred', 'user_ns')])},
              {'call': 'setitimer',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'ioprio_get',
               'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'vfork',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'setuid',
               'reason': set([('cred', 'euid'),
                                ('cred', 'suid'),
                                ('cred', 'uid'),
                                ('cred', 'user_ns')])},
              {'call': 'prctl',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'move_pages',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
              {'call': 'setpriority',
               'reason': set([('cred', 'user_ns'),
                                ('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'clone',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])},
              {'call': 'sched_getparam',
               'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'group_leader'),
                                ('task_struct', 'real_cred'),
                                ('task_struct', 'sighand')])}],
 'pselect6': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
```



```
              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_unlink',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'swapoff',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'sched_getaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sched_setparam',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchmod',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'memfd_create',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'ioprio_set',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'readlinkat',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'io_getevents',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchown',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_timedreceive',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'utime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'sched_setaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'semtimedop',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'settimeofday',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'timerfd_gettime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'uselib',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sigaltstack',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sched_setattr',
```

```
10950                'reason': set([('mm_segment_t', 'seg')])},
10951               {'call': 'migrate_pages',
10952                'reason': set([('mm_segment_t', 'seg')])},
10953               {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
10954               {'call': 'fchmodat',
10955                'reason': set([('timespec', 'tv_nsec'),
10956                               ('timespec', 'tv_sec')])},
10957               {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
10958               {'call': 'inotify_add_watch',
10959                'reason': set([('timespec', 'tv_nsec'),
10960                               ('timespec', 'tv_sec')])},
10961               {'call': 'timer_settime',
10962                'reason': set([('timespec', 'tv_nsec'),
10963                               ('timespec', 'tv_sec')])},
10964               {'call': 'ftruncate',
10965                'reason': set([('timespec', 'tv_nsec'),
10966                               ('timespec', 'tv_sec')])},
10967               {'call': 'timer_gettime',
10968                'reason': set([('timespec', 'tv_nsec'),
10969                               ('timespec', 'tv_sec')])},
10970               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
10971               {'call': 'ioctl',
10972                'reason': set([('timespec', 'tv_nsec'),
10973                               ('timespec', 'tv_sec')])},
10974               {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
10975               {'call': 'perf_event_open',
10976                'reason': set([('mm_segment_t', 'seg')])},
10977               {'call': 'linkat',
10978                'reason': set([('timespec', 'tv_nsec'),
10979                               ('timespec', 'tv_sec')])},
10980               {'call': 'stime',
10981                'reason': set([('timespec', 'tv_nsec'),
10982                               ('timespec', 'tv_sec')])},
10983               {'call': 'rt_sigaction',
10984                'reason': set([('mm_segment_t', 'seg')])},
10985               {'call': 'futimesat',
10986                'reason': set([('timespec', 'tv_nsec'),
10987                               ('timespec', 'tv_sec')])},
10988               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
10989               {'call': 'poll',
10990                'reason': set([('timespec', 'tv_nsec'),
10991                               ('timespec', 'tv_sec')])},
10992               {'call': 'select',
10993                'reason': set([('compat_timespec', 'tv_nsec'),
10994                               ('compat_timespec', 'tv_sec'),
10995                               ('timespec', 'tv_nsec'),
10996                               ('timespec', 'tv_sec')])},
10997               {'call': 'unlink',
10998                'reason': set([('timespec', 'tv_nsec'),
10999                               ('timespec', 'tv_sec')])},
11000               {'call': 'getpriority',
11001                'reason': set([('mm_segment_t', 'seg')])},
11002               {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
11003               {'call': 'nanosleep',
11004                'reason': set([('compat_timespec', 'tv_nsec'),
11005                               ('compat_timespec', 'tv_sec'),
11006                               ('timespec', 'tv_nsec'),
11007                               ('timespec', 'tv_sec')])},
11008               {'call': 'mq_getsetattr',
11009                'reason': set([('timespec', 'tv_nsec'),
11010                               ('timespec', 'tv_sec')])},
11011               {'call': 'faccessat',
11012                'reason': set([('timespec', 'tv_nsec'),
11013                               ('timespec', 'tv_sec')])},
11014               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
11015               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
```



```
11016               {'call': 'get_robust_list',
11017                'reason': set([('mm_segment_t', 'seg')])},
11018               {'call': 'mq_timedsend',
11019                'reason': set([('mm_segment_t', 'seg'),
11020                               ('timespec', 'tv_nsec'),
11021                               ('timespec', 'tv_sec')])},
11022               {'call': 'sched_getscheduler',
11023                'reason': set([('mm_segment_t', 'seg')])},
11024               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
11025               {'call': 'swapon',
11026                'reason': set([('timespec', 'tv_nsec'),
11027                               ('timespec', 'tv_sec')])},
11028               {'call': 'epoll_wait',
11029                'reason': set([('timespec', 'tv_nsec'),
11030                               ('timespec', 'tv_sec')])},
11031               {'call': 'sched_getattr',
11032                'reason': set([('mm_segment_t', 'seg')])},
11033               {'call': 'fchownat',
11034                'reason': set([('timespec', 'tv_nsec'),
11035                               ('timespec', 'tv_sec')])},
11036               {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
11037               {'call': 'fstat',
11038                'reason': set([('timespec', 'tv_nsec'),
11039                               ('timespec', 'tv_sec')])},
11040               {'call': 'timerfd_settime',
11041                'reason': set([('timespec', 'tv_nsec'),
11042                               ('timespec', 'tv_sec')])},
11043               {'call': 'sched_setscheduler',
11044                'reason': set([('mm_segment_t', 'seg')])},
11045               {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
11046               {'call': 'ioprio_get',
11047                'reason': set([('mm_segment_t', 'seg')])},
11048               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
11049               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
11050               {'call': 'move_pages',
11051                'reason': set([('mm_segment_t', 'seg')])},
11052               {'call': 'setpriority',
11053                'reason': set([('mm_segment_t', 'seg')])},
11054               {'call': 'mq_notify',
11055                'reason': set([('timespec', 'tv_nsec'),
11056                               ('timespec', 'tv_sec')])},
11057               {'call': 'sendfile',
11058                'reason': set([('timespec', 'tv_nsec'),
11059                               ('timespec', 'tv_sec')])},
11060               {'call': 'newfstat',
11061                'reason': set([('timespec', 'tv_nsec'),
11062                               ('timespec', 'tv_sec')])},
11063               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
11064               {'call': 'clock_nanosleep',
11065                'reason': set([('compat_timespec', 'tv_nsec'),
11066                               ('compat_timespec', 'tv_sec'),
11067                               ('timespec', 'tv_nsec'),
11068                               ('timespec', 'tv_sec')])},
11069               {'call': 'unlinkat',
11070                'reason': set([('timespec', 'tv_nsec'),
11071                               ('timespec', 'tv_sec')])},
11072               {'call': 'sched_getparam',
11073                'reason': set([('mm_segment_t', 'seg')])},
11074               {'call': 'futex',
11075                'reason': set([('timespec', 'tv_nsec'),
11076                               ('timespec', 'tv_sec')])},
11077               {'call': 'recvmmsg',
11078                'reason': set([('timespec', 'tv_nsec'),
11079                               ('timespec', 'tv_sec')])},
11080               {'call': 'sendfile64',
11081                'reason': set([('timespec', 'tv_nsec'),
```

```
                              ('timespec', 'tv_sec')])},
              {'call': 'ppoll',
               'reason': set([('compat_timespec', 'tv_nsec'),
                              ('compat_timespec', 'tv_sec'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])}],
 'ptrace': [{'call': 'keyctl',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'msgrcv',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'kill',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'pause', 'reason': set([('task_struct', 'state')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'ioprio_set',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'getppid',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
```



```
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'capget',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_setaffinity',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'signal',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'semtimedop',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'umount',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'setsid',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'real_parent'),
                            ('task_struct', 'state')])},
            {'call': 'sigaltstack',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'parent'),
                            ('task_struct', 'ptrace'),
```

```
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'getitimer',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'setpgid',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'rt_sigsuspend',
             'reason': set([('task_struct', 'state')])},
            {'call': 'getsid',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'prlimit64',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'perf_event_open',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'getpgid',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
```



```
                              ('task_struct', 'state')])},
            {'call': 'getpriority',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'sigaction',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'setns',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'fork',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'epoll_wait', 'reason': set([('task_struct', 'state')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'getrusage',
             'reason': set([('task_struct', 'exit_state'),
                              ('task_struct', 'flags'),
                              ('task_struct', 'parent'),
                              ('task_struct', 'ptrace'),
                              ('task_struct', 'real_parent'),
                              ('task_struct', 'state')])},
            {'call': 'sched_setscheduler',
```

```
11346              'reason': set([('task_struct', 'exit_state'),
11347                             ('task_struct', 'flags'),
11348                             ('task_struct', 'parent'),
11349                             ('task_struct', 'ptrace'),
11350                             ('task_struct', 'real_parent'),
11351                             ('task_struct', 'state')])},
11352             {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
11353             {'call': 'setitimer',
11354              'reason': set([('task_struct', 'exit_state'),
11355                             ('task_struct', 'flags'),
11356                             ('task_struct', 'parent'),
11357                             ('task_struct', 'ptrace'),
11358                             ('task_struct', 'real_parent'),
11359                             ('task_struct', 'state')])},
11360             {'call': 'ioprio_get',
11361              'reason': set([('task_struct', 'exit_state'),
11362                             ('task_struct', 'flags'),
11363                             ('task_struct', 'parent'),
11364                             ('task_struct', 'ptrace'),
11365                             ('task_struct', 'real_parent'),
11366                             ('task_struct', 'state')])},
11367             {'call': 'vfork',
11368              'reason': set([('task_struct', 'exit_state'),
11369                             ('task_struct', 'flags'),
11370                             ('task_struct', 'parent'),
11371                             ('task_struct', 'ptrace'),
11372                             ('task_struct', 'real_parent'),
11373                             ('task_struct', 'state')])},
11374             {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
11375             {'call': 'prctl',
11376              'reason': set([('task_struct', 'exit_state'),
11377                             ('task_struct', 'flags'),
11378                             ('task_struct', 'parent'),
11379                             ('task_struct', 'ptrace'),
11380                             ('task_struct', 'real_parent'),
11381                             ('task_struct', 'state')])},
11382             {'call': 'move_pages',
11383              'reason': set([('task_struct', 'exit_state'),
11384                             ('task_struct', 'flags'),
11385                             ('task_struct', 'parent'),
11386                             ('task_struct', 'ptrace'),
11387                             ('task_struct', 'real_parent'),
11388                             ('task_struct', 'state')])},
11389             {'call': 'setpriority',
11390              'reason': set([('task_struct', 'exit_state'),
11391                             ('task_struct', 'flags'),
11392                             ('task_struct', 'parent'),
11393                             ('task_struct', 'ptrace'),
11394                             ('task_struct', 'real_parent'),
11395                             ('task_struct', 'state')])},
11396             {'call': 'clone',
11397              'reason': set([('task_struct', 'exit_state'),
11398                             ('task_struct', 'flags'),
11399                             ('task_struct', 'parent'),
11400                             ('task_struct', 'ptrace'),
11401                             ('task_struct', 'real_parent'),
11402                             ('task_struct', 'state')])},
11403             {'call': 'sigsuspend', 'reason': set([('task_struct', 'state')])},
11404             {'call': 'sched_getparam',
11405              'reason': set([('task_struct', 'exit_state'),
11406                             ('task_struct', 'flags'),
11407                             ('task_struct', 'parent'),
11408                             ('task_struct', 'ptrace'),
11409                             ('task_struct', 'real_parent'),
11410                             ('task_struct', 'state')])}],
11411  'pwrite64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
```



```
11412             {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
11413             {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
11414             {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
11415             {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
11416             {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
11417             {'call': 'readahead', 'reason': set([('fd', 'flags')])},
11418             {'call': 'getdents', 'reason': set([('fd', 'flags')])},
11419             {'call': 'writev', 'reason': set([('fd', 'flags')])},
11420             {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
11421             {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
11422             {'call': 'pread64', 'reason': set([('fd', 'flags')])},
11423             {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
11424             {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
11425             {'call': 'remap_file_pages',
11426              'reason': set([('file', 'f_mode')])},
11427             {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
11428             {'call': 'read', 'reason': set([('fd', 'flags')])},
11429             {'call': 'fchown', 'reason': set([('fd', 'flags')])},
11430             {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
11431             {'call': 'utime', 'reason': set([('fd', 'flags')])},
11432             {'call': 'fsync', 'reason': set([('fd', 'flags')])},
11433             {'call': 'bpf', 'reason': set([('fd', 'flags')])},
11434             {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
11435             {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
11436             {'call': 'sendto', 'reason': set([('fd', 'flags')])},
11437             {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
11438             {'call': 'tee', 'reason': set([('fd', 'flags')])},
11439             {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
11440             {'call': 'lseek', 'reason': set([('fd', 'flags')])},
11441             {'call': 'connect', 'reason': set([('fd', 'flags')])},
11442             {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
11443             {'call': 'epoll_ctl',
11444              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
11445             {'call': 'flock',
11446              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
11447             {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
11448             {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
11449             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
11450             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
11451             {'call': 'accept4',
11452              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
11453             {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
11454             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
11455             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
11456             {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
11457             {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
11458             {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
11459             {'call': 'splice', 'reason': set([('fd', 'flags')])},
11460             {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
11461             {'call': 'preadv', 'reason': set([('fd', 'flags')])},
11462             {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
11463             {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
11464             {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
11465             {'call': 'socket', 'reason': set([('file', 'f_mode')])},
11466             {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
11467             {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
11468             {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
11469             {'call': 'perf_event_open',
11470              'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
11471             {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
11472             {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
11473             {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
11474             {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
11475             {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
11476             {'call': 'acct', 'reason': set([('file', 'f_mode')])},
11477             {'call': 'open', 'reason': set([('file', 'f_mode')])},
```

```
11478               {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
11479               {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
11480               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
11481               {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
11482               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
11483               {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
11484               {'call': 'listen', 'reason': set([('fd', 'flags')])},
11485               {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
11486               {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
11487               {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
11488               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
11489               {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
11490               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
11491               {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
11492               {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
11493               {'call': 'llseek', 'reason': set([('fd', 'flags')])},
11494               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
11495               {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
11496               {'call': 'readv', 'reason': set([('fd', 'flags')])},
11497               {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
11498               {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
11499               {'call': 'write', 'reason': set([('fd', 'flags')])},
11500               {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
11501               {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
11502               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
11503               {'call': 'open_by_handle_at',
11504                'reason': set([('file', 'f_mode')])},
11505               {'call': 'bind', 'reason': set([('fd', 'flags')])},
11506               {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
11507               {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
11508  'pwritev': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
11509              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
11510              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
11511              {'call': 'remap_file_pages',
11512               'reason': set([('file', 'f_mode')])},
11513              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
11514              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
11515              {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
11516              {'call': 'flock', 'reason': set([('file', 'f_mode')])},
11517              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
11518              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
11519              {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
11520              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
11521              {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
11522              {'call': 'socket', 'reason': set([('file', 'f_mode')])},
11523              {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
11524              {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
11525              {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
11526              {'call': 'acct', 'reason': set([('file', 'f_mode')])},
11527              {'call': 'open', 'reason': set([('file', 'f_mode')])},
11528              {'call': 'dup', 'reason': set([('file', 'f_mode')])},
11529              {'call': 'setns', 'reason': set([('file', 'f_mode')])},
11530              {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
11531              {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
11532              {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
11533              {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
11534              {'call': 'open_by_handle_at',
11535               'reason': set([('file', 'f_mode')])}],
11536  'pwritev2': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
11537               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
11538               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
11539               {'call': 'remap_file_pages',
11540                'reason': set([('file', 'f_mode')])},
11541               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
11542               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
11543               {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
```



```
11544               {'call': 'flock', 'reason': set([('file', 'f_mode')])},
11545               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
11546               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
11547               {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
11548               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
11549               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
11550               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
11551               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
11552               {'call': 'perf_event_open',
11553                'reason': set([('file', 'f_mode')])},
11554               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
11555               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
11556               {'call': 'open', 'reason': set([('file', 'f_mode')])},
11557               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
11558               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
11559               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
11560               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
11561               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
11562               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
11563               {'call': 'open_by_handle_at',
11564                'reason': set([('file', 'f_mode')])}],
11565  'pwritev64': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
11566                {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
11567                {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
11568                {'call': 'remap_file_pages',
11569                 'reason': set([('file', 'f_mode')])},
11570                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
11571                {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
11572                {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
11573                {'call': 'flock', 'reason': set([('file', 'f_mode')])},
11574                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
11575                {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
11576                {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
11577                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
11578                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
11579                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
11580                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
11581                {'call': 'perf_event_open',
11582                 'reason': set([('file', 'f_mode')])},
11583                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
11584                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
11585                {'call': 'open', 'reason': set([('file', 'f_mode')])},
11586                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
11587                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
11588                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
11589                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
11590                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
11591                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
11592                {'call': 'open_by_handle_at',
11593                 'reason': set([('file', 'f_mode')])}],
11594  'pwritev64v2': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
11595                  {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
11596                  {'call': 'memfd_create',
11597                   'reason': set([('file', 'f_mode')])},
11598                  {'call': 'remap_file_pages',
11599                   'reason': set([('file', 'f_mode')])},
11600                  {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
11601                  {'call': 'epoll_create1',
11602                   'reason': set([('file', 'f_mode')])},
11603                  {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
11604                  {'call': 'flock', 'reason': set([('file', 'f_mode')])},
11605                  {'call': 'openat', 'reason': set([('file', 'f_mode')])},
11606                  {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
11607                  {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
11608                  {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
11609                  {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
```

```
                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
                {'call': 'perf_event_open',
                 'reason': set([('file', 'f_mode')])},
                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
                {'call': 'open', 'reason': set([('file', 'f_mode')])},
                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_mode')])}],
 'quotactl': [{'call': 'syncfs',
               'reason': set([('super_block', 's_flags'),
                              ('super_block', 's_qcop'),
                              ('super_block', 's_quota_types')])},
              {'call': 'sysfs', 'reason': set([('filename', 'name')])},
              {'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
              {'call': 'swapoff', 'reason': set([('filename', 'name')])},
              {'call': 'ustat',
               'reason': set([('super_block', 's_flags'),
                              ('super_block', 's_qcop'),
                              ('super_block', 's_quota_types')])},
              {'call': 'umount',
               'reason': set([('super_block', 's_flags'),
                              ('super_block', 's_qcop'),
                              ('super_block', 's_quota_types')])},
              {'call': 'openat', 'reason': set([('filename', 'name')])},
              {'call': 'uselib', 'reason': set([('filename', 'name')])},
              {'call': 'renameat2', 'reason': set([('filename', 'name')])},
              {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
              {'call': 'acct', 'reason': set([('filename', 'name')])},
              {'call': 'open', 'reason': set([('filename', 'name')])},
              {'call': 'unlink', 'reason': set([('filename', 'name')])},
              {'call': 'rmdir', 'reason': set([('filename', 'name')])},
              {'call': 'swapon',
               'reason': set([('filename', 'name'),
                              ('super_block', 's_flags'),
                              ('super_block', 's_qcop'),
                              ('super_block', 's_quota_types')])},
              {'call': 'mq_open', 'reason': set([('filename', 'name')])},
              {'call': 'unlinkat', 'reason': set([('filename', 'name')])}],
 'read': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
          {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
          {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
          {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
          {'call': 'readahead', 'reason': set([('fd', 'flags')])},
          {'call': 'getdents', 'reason': set([('fd', 'flags')])},
          {'call': 'writev', 'reason': set([('fd', 'flags')])},
          {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
          {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
          {'call': 'pread64', 'reason': set([('fd', 'flags')])},
          {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
          {'call': 'fchown', 'reason': set([('fd', 'flags')])},
          {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
          {'call': 'utime', 'reason': set([('fd', 'flags')])},
          {'call': 'fsync', 'reason': set([('fd', 'flags')])},
          {'call': 'bpf', 'reason': set([('fd', 'flags')])},
          {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
          {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
          {'call': 'sendto', 'reason': set([('fd', 'flags')])},
          {'call': 'tee', 'reason': set([('fd', 'flags')])},
          {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
```



```
          {'call': 'lseek', 'reason': set([('fd', 'flags')])},
          {'call': 'connect', 'reason': set([('fd', 'flags')])},
          {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
          {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
          {'call': 'flock', 'reason': set([('fd', 'flags')])},
          {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
          {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
          {'call': 'accept4', 'reason': set([('fd', 'flags')])},
          {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
          {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
          {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
          {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
          {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
          {'call': 'splice', 'reason': set([('fd', 'flags')])},
          {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
          {'call': 'preadv', 'reason': set([('fd', 'flags')])},
          {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
          {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
          {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
          {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
          {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
          {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
          {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
          {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
          {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
          {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
          {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
          {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
          {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
          {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
          {'call': 'listen', 'reason': set([('fd', 'flags')])},
          {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
          {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
          {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
          {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
          {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
          {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
          {'call': 'llseek', 'reason': set([('fd', 'flags')])},
          {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
          {'call': 'readv', 'reason': set([('fd', 'flags')])},
          {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
          {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
          {'call': 'write', 'reason': set([('fd', 'flags')])},
          {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
          {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
          {'call': 'bind', 'reason': set([('fd', 'flags')])},
          {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
          {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'readahead': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
               {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
               {'call': 'mq_unlink',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
               {'call': 'swapoff',
                'reason': set([('address_space', 'a_ops'),
                               ('file', 'f_mode')])},
               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
               {'call': 'writev', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
               {'call': 'fchmod',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'pread64', 'reason': set([('fd', 'flags')])},
               {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
               {'call': 'memfd_create',
```

```
                 'reason': set([('address_space', 'a_ops'),
                                ('file', 'f_mode')])},
               {'call': 'remap_file_pages',
                'reason': set([('file', 'f_mode')])},
               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
               {'call': 'readlinkat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'read', 'reason': set([('fd', 'flags')])},
               {'call': 'fchown',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'mq_timedreceive',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'utime', 'reason': set([('fd', 'flags')])},
               {'call': 'fsync', 'reason': set([('fd', 'flags')])},
               {'call': 'bpf', 'reason': set([('fd', 'flags')])},
               {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
               {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'sendto', 'reason': set([('fd', 'flags')])},
               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
               {'call': 'tee', 'reason': set([('fd', 'flags')])},
               {'call': 'sync_file_range',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'lseek', 'reason': set([('fd', 'flags')])},
               {'call': 'connect', 'reason': set([('fd', 'flags')])},
               {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
               {'call': 'epoll_ctl',
                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
               {'call': 'flock',
                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
               {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
               {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
               {'call': 'uselib',
                'reason': set([('address_space', 'a_ops'),
                               ('file', 'f_mode')])},
               {'call': 'accept4',
                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
               {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
               {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
               {'call': 'fchmodat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'inotify_add_watch',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
               {'call': 'splice', 'reason': set([('fd', 'flags')])},
               {'call': 'ftruncate',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'preadv', 'reason': set([('fd', 'flags')])},
               {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
               {'call': 'shmat',
                'reason': set([('address_space', 'a_ops'),
                               ('file', 'f_mode')])},
               {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
               {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
               {'call': 'ioctl',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
               {'call': 'perf_event_open',
                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
               {'call': 'linkat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
```



```
               {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
               {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
               {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
               {'call': 'open', 'reason': set([('file', 'f_mode')])},
               {'call': 'unlink',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_getsetattr',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'faccessat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
               {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
               {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
               {'call': 'listen', 'reason': set([('fd', 'flags')])},
               {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_timedsend',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
               {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
               {'call': 'swapon',
                'reason': set([('address_space', 'a_ops'),
                               ('file', 'f_mode')])},
               {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
               {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
               {'call': 'fchownat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'llseek', 'reason': set([('fd', 'flags')])},
               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
               {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
               {'call': 'readv', 'reason': set([('fd', 'flags')])},
               {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
               {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
               {'call': 'write', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_notify',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'sendfile',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])},
               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
               {'call': 'unlinkat',
                'reason': set([('address_space', 'a_ops')])},
               {'call': 'open_by_handle_at',
                'reason': set([('file', 'f_mode')])},
               {'call': 'bind', 'reason': set([('fd', 'flags')])},
               {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'sendfile64',
                'reason': set([('address_space', 'a_ops'), ('fd', 'flags')])}],
 'readlinkat': [{'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
                {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
                {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
                {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
                {'call': 'remap_file_pages',
                 'reason': set([('path', 'dentry')])},
                {'call': 'dup3', 'reason': set([('path', 'dentry')])},
                {'call': 'unshare', 'reason': set([('path', 'dentry')])},
                {'call': 'epoll_create1',
                 'reason': set([('path', 'dentry')])},
                {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
                {'call': 'flock', 'reason': set([('path', 'dentry')])},
                {'call': 'openat', 'reason': set([('path', 'dentry')])},
                {'call': 'lookup_dcookie',
                 'reason': set([('path', 'dentry')])},
```

```
                {'call': 'uselib', 'reason': set([('path', 'dentry')])},
                {'call': 'accept4', 'reason': set([('path', 'dentry')])},
                {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
                {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
                {'call': 'shmat', 'reason': set([('path', 'dentry')])},
                {'call': 'socket', 'reason': set([('path', 'dentry')])},
                {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
                {'call': 'perf_event_open',
                 'reason': set([('path', 'dentry')])},
                {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
                {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
                {'call': 'acct', 'reason': set([('path', 'dentry')])},
                {'call': 'open', 'reason': set([('path', 'dentry')])},
                {'call': 'dup', 'reason': set([('path', 'dentry')])},
                {'call': 'setns', 'reason': set([('path', 'dentry')])},
                {'call': 'shmctl', 'reason': set([('path', 'dentry')])},
                {'call': 'swapon', 'reason': set([('path', 'dentry')])},
                {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
                {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('path', 'dentry')])}],
 'reboot': [{'call': 'perf_event_open',
             'reason': set([('pid_namespace', 'user_ns')])},
            {'call': 'acct', 'reason': set([('pid_namespace', 'user_ns')])},
            {'call': 'setns', 'reason': set([('pid_namespace', 'user_ns')])}],
 'recvfrom': [{'call': 'eventfd2', 'reason': set([('file', 'f_flags')])},
              {'call': 'swapoff', 'reason': set([('file', 'f_flags')])},
              {'call': 'memfd_create', 'reason': set([('file', 'f_flags')])},
              {'call': 'remap_file_pages',
               'reason': set([('file', 'f_flags')])},
              {'call': 'dup3', 'reason': set([('file', 'f_flags')])},
              {'call': 'epoll_create1', 'reason': set([('file', 'f_flags')])},
              {'call': 'epoll_ctl', 'reason': set([('file', 'f_flags')])},
              {'call': 'flock', 'reason': set([('file', 'f_flags')])},
              {'call': 'openat', 'reason': set([('file', 'f_flags')])},
              {'call': 'uselib', 'reason': set([('file', 'f_flags')])},
              {'call': 'accept4', 'reason': set([('file', 'f_flags')])},
              {'call': 'socketpair', 'reason': set([('file', 'f_flags')])},
              {'call': 'shmat', 'reason': set([('file', 'f_flags')])},
              {'call': 'socket', 'reason': set([('file', 'f_flags')])},
              {'call': 'pipe2', 'reason': set([('file', 'f_flags')])},
              {'call': 'perf_event_open',
               'reason': set([('file', 'f_flags')])},
              {'call': 'shmdt', 'reason': set([('file', 'f_flags')])},
              {'call': 'acct', 'reason': set([('file', 'f_flags')])},
              {'call': 'open', 'reason': set([('file', 'f_flags')])},
              {'call': 'mq_getsetattr', 'reason': set([('file', 'f_flags')])},
              {'call': 'dup', 'reason': set([('file', 'f_flags')])},
              {'call': 'setns', 'reason': set([('file', 'f_flags')])},
              {'call': 'shmctl', 'reason': set([('file', 'f_flags')])},
              {'call': 'swapon', 'reason': set([('file', 'f_flags')])},
              {'call': 'mmap_pgoff', 'reason': set([('file', 'f_flags')])},
              {'call': 'mq_open', 'reason': set([('file', 'f_flags')])},
              {'call': 'open_by_handle_at',
               'reason': set([('file', 'f_flags')])}],
 'recvmmsg': [{'call': 'rt_sigtimedwait',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'mq_unlink',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'swapoff',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'fchmod',
               'reason': set([('timespec', 'tv_nsec'),
```



```
                              ('timespec', 'tv_sec')])},
              {'call': 'memfd_create',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'readlinkat',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'io_getevents',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'fchown',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'mq_timedreceive',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'utime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'semtimedop',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'recvfrom',
               'reason': set([('msghdr', 'msg_flags'),
                              ('socket', 'file'),
                              ('socket', 'sk')])},
              {'call': 'settimeofday',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'sendto',
               'reason': set([('msghdr', 'msg_flags'),
                              ('socket', 'file'),
                              ('socket', 'sk')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'timerfd_gettime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'connect',
               'reason': set([('socket', 'file'), ('socket', 'sk')])},
              {'call': 'getsockname',
               'reason': set([('socket', 'file'), ('socket', 'sk')])},
              {'call': 'pselect6',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'uselib',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'accept4',
               'reason': set([('socket', 'file'), ('socket', 'sk')])},
              {'call': 'fchmodat',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'inotify_add_watch',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'timer_settime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'ftruncate',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'timer_gettime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
```

```
12006                {'call': 'getpeername',
12007                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12008                {'call': 'setsockopt',
12009                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12010                {'call': 'ioctl',
12011                 'reason': set([('timespec', 'tv_nsec'),
12012                                ('timespec', 'tv_sec')])},
12013                {'call': 'linkat',
12014                 'reason': set([('timespec', 'tv_nsec'),
12015                                ('timespec', 'tv_sec')])},
12016                {'call': 'sendmsg',
12017                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12018                {'call': 'stime',
12019                 'reason': set([('timespec', 'tv_nsec'),
12020                                ('timespec', 'tv_sec')])},
12021                {'call': 'futimesat',
12022                 'reason': set([('timespec', 'tv_nsec'),
12023                                ('timespec', 'tv_sec')])},
12024                {'call': 'shutdown',
12025                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12026                {'call': 'poll',
12027                 'reason': set([('timespec', 'tv_nsec'),
12028                                ('timespec', 'tv_sec')])},
12029                {'call': 'select',
12030                 'reason': set([('timespec', 'tv_nsec'),
12031                                ('timespec', 'tv_sec')])},
12032                {'call': 'unlink',
12033                 'reason': set([('timespec', 'tv_nsec'),
12034                                ('timespec', 'tv_sec')])},
12035                {'call': 'getsockopt',
12036                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12037                {'call': 'nanosleep',
12038                 'reason': set([('timespec', 'tv_nsec'),
12039                                ('timespec', 'tv_sec')])},
12040                {'call': 'mq_getsetattr',
12041                 'reason': set([('timespec', 'tv_nsec'),
12042                                ('timespec', 'tv_sec')])},
12043                {'call': 'faccessat',
12044                 'reason': set([('timespec', 'tv_nsec'),
12045                                ('timespec', 'tv_sec')])},
12046                {'call': 'listen',
12047                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12048                {'call': 'mq_timedsend',
12049                 'reason': set([('timespec', 'tv_nsec'),
12050                                ('timespec', 'tv_sec')])},
12051                {'call': 'swapon',
12052                 'reason': set([('timespec', 'tv_nsec'),
12053                                ('timespec', 'tv_sec')])},
12054                {'call': 'epoll_wait',
12055                 'reason': set([('timespec', 'tv_nsec'),
12056                                ('timespec', 'tv_sec')])},
12057                {'call': 'fchownat',
12058                 'reason': set([('timespec', 'tv_nsec'),
12059                                ('timespec', 'tv_sec')])},
12060                {'call': 'fstat',
12061                 'reason': set([('timespec', 'tv_nsec'),
12062                                ('timespec', 'tv_sec')])},
12063                {'call': 'timerfd_settime',
12064                 'reason': set([('timespec', 'tv_nsec'),
12065                                ('timespec', 'tv_sec')])},
12066                {'call': 'recvmsg',
12067                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12068                {'call': 'mq_notify',
12069                 'reason': set([('timespec', 'tv_nsec'),
12070                                ('timespec', 'tv_sec')])},
12071                {'call': 'sendfile',
```



```
12072                 'reason': set([('timespec', 'tv_nsec'),
12073                                ('timespec', 'tv_sec')])},
12074                {'call': 'sendmmsg',
12075                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12076                {'call': 'newfstat',
12077                 'reason': set([('timespec', 'tv_nsec'),
12078                                ('timespec', 'tv_sec')])},
12079                {'call': 'clock_nanosleep',
12080                 'reason': set([('timespec', 'tv_nsec'),
12081                                ('timespec', 'tv_sec')])},
12082                {'call': 'unlinkat',
12083                 'reason': set([('timespec', 'tv_nsec'),
12084                                ('timespec', 'tv_sec')])},
12085                {'call': 'bind',
12086                 'reason': set([('socket', 'file'), ('socket', 'sk')])},
12087                {'call': 'futex',
12088                 'reason': set([('timespec', 'tv_nsec'),
12089                                ('timespec', 'tv_sec')])},
12090                {'call': 'sendfile64',
12091                 'reason': set([('timespec', 'tv_nsec'),
12092                                ('timespec', 'tv_sec')])},
12093                {'call': 'ppoll',
12094                 'reason': set([('timespec', 'tv_nsec'),
12095                                ('timespec', 'tv_sec')])}],
12096  'recvmsg': [{'call': 'recvfrom', 'reason': set([('socket', 'file')])},
12097              {'call': 'sendto', 'reason': set([('socket', 'file')])},
12098              {'call': 'connect', 'reason': set([('socket', 'file')])},
12099              {'call': 'getsockname', 'reason': set([('socket', 'file')])},
12100              {'call': 'accept4', 'reason': set([('socket', 'file')])},
12101              {'call': 'getpeername', 'reason': set([('socket', 'file')])},
12102              {'call': 'setsockopt', 'reason': set([('socket', 'file')])},
12103              {'call': 'sendmsg', 'reason': set([('socket', 'file')])},
12104              {'call': 'shutdown', 'reason': set([('socket', 'file')])},
12105              {'call': 'getsockopt', 'reason': set([('socket', 'file')])},
12106              {'call': 'listen', 'reason': set([('socket', 'file')])},
12107              {'call': 'sendmmsg', 'reason': set([('socket', 'file')])},
12108              {'call': 'bind', 'reason': set([('socket', 'file')])},
12109              {'call': 'recvmmsg', 'reason': set([('socket', 'file')])}],
12110  'remap_file_pages': [{'call': 'shmdt',
12111                        'reason': set([('vm_area_struct', 'vm_end'),
12112                                       ('vm_area_struct', 'vm_file'),
12113                                       ('vm_area_struct', 'vm_flags'),
12114                                       ('vm_area_struct', 'vm_start')])},
12115                       {'call': 'brk',
12116                        'reason': set([('vm_area_struct', 'vm_end'),
12117                                       ('vm_area_struct', 'vm_file'),
12118                                       ('vm_area_struct', 'vm_flags'),
12119                                       ('vm_area_struct', 'vm_start')])},
12120                       {'call': 'get_mempolicy',
12121                        'reason': set([('vm_area_struct', 'vm_end'),
12122                                       ('vm_area_struct', 'vm_file'),
12123                                       ('vm_area_struct', 'vm_flags'),
12124                                       ('vm_area_struct', 'vm_start')])},
12125                       {'call': 'munlockall',
12126                        'reason': set([('vm_area_struct', 'vm_end'),
12127                                       ('vm_area_struct', 'vm_file'),
12128                                       ('vm_area_struct', 'vm_flags'),
12129                                       ('vm_area_struct', 'vm_start')])},
12130                       {'call': 'pkey_mprotect',
12131                        'reason': set([('vm_area_struct', 'vm_end'),
12132                                       ('vm_area_struct', 'vm_file'),
12133                                       ('vm_area_struct', 'vm_flags'),
12134                                       ('vm_area_struct', 'vm_start')])},
12135                       {'call': 'madvise',
12136                        'reason': set([('vm_area_struct', 'vm_end'),
12137                                       ('vm_area_struct', 'vm_file'),
```

```
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])},
                        {'call': 'mprotect',
                         'reason': set([('vm_area_struct', 'vm_end'),
                                        ('vm_area_struct', 'vm_file'),
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])},
                        {'call': 'mremap',
                         'reason': set([('vm_area_struct', 'vm_end'),
                                        ('vm_area_struct', 'vm_file'),
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])},
                        {'call': 'prctl',
                         'reason': set([('vm_area_struct', 'vm_end'),
                                        ('vm_area_struct', 'vm_file'),
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])},
                        {'call': 'munlock',
                         'reason': set([('vm_area_struct', 'vm_end'),
                                        ('vm_area_struct', 'vm_file'),
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])},
                        {'call': 'mincore',
                         'reason': set([('vm_area_struct', 'vm_end'),
                                        ('vm_area_struct', 'vm_file'),
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])},
                        {'call': 'mlockall',
                         'reason': set([('vm_area_struct', 'vm_end'),
                                        ('vm_area_struct', 'vm_file'),
                                        ('vm_area_struct', 'vm_flags'),
                                        ('vm_area_struct', 'vm_start')])}],
 'removexattr': [{'call': 'eventfd2',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'swapoff',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'pivot_root',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'memfd_create',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'remap_file_pages',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'dup3',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'unshare',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'epoll_create1',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'epoll_ctl',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'flock',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'openat',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'lookup_dcookie',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'uselib',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'accept4',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'socketpair',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'getcwd',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'shmat',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```



```
                 {'call': 'socket',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'pipe2',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'perf_event_open',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'shmdt',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'quotactl',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'acct',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'open',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'dup',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'setns',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'shmctl',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'swapon',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'mmap_pgoff',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'mq_open',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])},
                 {'call': 'open_by_handle_at',
                  'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
 'renameat2': [{'call': 'sysfs',
                'reason': set([('filename', 'name'),
                               ('filename', 'refcnt')])},
               {'call': 'eventfd2',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mq_unlink',
                'reason': set([('dentry', 'd_inode'),
                               ('filename', 'name'),
                               ('filename', 'refcnt')])},
               {'call': 'swapoff',
                'reason': set([('filename', 'name'),
                               ('filename', 'refcnt'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'pivot_root',
                'reason': set([('dentry', 'd_inode'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
               {'call': 'memfd_create',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'remap_file_pages',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'dup3',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'unshare',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
               {'call': 'epoll_create1',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'epoll_ctl',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'flock',
                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
               {'call': 'openat',
                'reason': set([('filename', 'name'),
                               ('filename', 'refcnt'),
                               ('path', 'dentry'),
                               ('path', 'mnt')])},
```

```
12270                {'call': 'lookup_dcookie',
12271                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12272                {'call': 'uselib',
12273                 'reason': set([('filename', 'name'),
12274                                ('filename', 'refcnt'),
12275                                ('path', 'dentry'),
12276                                ('path', 'mnt')])},
12277                {'call': 'accept4',
12278                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12279                {'call': 'socketpair',
12280                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12281                {'call': 'getcwd',
12282                 'reason': set([('dentry', 'd_inode'),
12283                                ('path', 'dentry'),
12284                                ('path', 'mnt')])},
12285                {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
12286                {'call': 'shmat',
12287                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12288                {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
12289                {'call': 'socket',
12290                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12291                {'call': 'symlinkat',
12292                 'reason': set([('dentry', 'd_inode'),
12293                                ('filename', 'name'),
12294                                ('filename', 'refcnt')])},
12295                {'call': 'pipe2',
12296                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12297                {'call': 'perf_event_open',
12298                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12299                {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
12300                {'call': 'shmdt',
12301                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12302                {'call': 'quotactl',
12303                 'reason': set([('filename', 'name'),
12304                                ('filename', 'refcnt'),
12305                                ('path', 'dentry'),
12306                                ('path', 'mnt')])},
12307                {'call': 'acct',
12308                 'reason': set([('filename', 'name'),
12309                                ('filename', 'refcnt'),
12310                                ('path', 'dentry'),
12311                                ('path', 'mnt')])},
12312                {'call': 'open',
12313                 'reason': set([('filename', 'name'),
12314                                ('filename', 'refcnt'),
12315                                ('path', 'dentry'),
12316                                ('path', 'mnt')])},
12317                {'call': 'unlink',
12318                 'reason': set([('dentry', 'd_inode'),
12319                                ('filename', 'name'),
12320                                ('filename', 'refcnt')])},
12321                {'call': 'rmdir',
12322                 'reason': set([('dentry', 'd_inode'),
12323                                ('filename', 'name'),
12324                                ('filename', 'refcnt')])},
12325                {'call': 'dup',
12326                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12327                {'call': 'setns',
12328                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12329                {'call': 'shmctl',
12330                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12331                {'call': 'swapon',
12332                 'reason': set([('filename', 'name'),
12333                                ('filename', 'refcnt'),
12334                                ('path', 'dentry'),
12335                                ('path', 'mnt')])},
```



```
12336                {'call': 'mmap_pgoff',
12337                 'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12338                {'call': 'mq_open',
12339                 'reason': set([('dentry', 'd_inode'),
12340                                ('filename', 'name'),
12341                                ('filename', 'refcnt'),
12342                                ('path', 'dentry'),
12343                                ('path', 'mnt')])},
12344                {'call': 'unlinkat',
12345                 'reason': set([('dentry', 'd_inode'),
12346                                ('filename', 'name'),
12347                                ('filename', 'refcnt')])},
12348                {'call': 'open_by_handle_at',
12349                 'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
12350  'rmdir': [{'call': 'eventfd2',
12351             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12352            {'call': 'mq_unlink', 'reason': set([('dentry', 'd_inode')])},
12353            {'call': 'swapoff',
12354             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12355            {'call': 'pivot_root',
12356             'reason': set([('dentry', 'd_inode'),
12357                            ('path', 'dentry'),
12358                            ('path', 'mnt')])},
12359            {'call': 'memfd_create',
12360             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12361            {'call': 'remap_file_pages',
12362             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12363            {'call': 'dup3',
12364             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12365            {'call': 'unshare',
12366             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12367            {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
12368            {'call': 'epoll_create1',
12369             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12370            {'call': 'epoll_ctl',
12371             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12372            {'call': 'flock',
12373             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12374            {'call': 'openat',
12375             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12376            {'call': 'lookup_dcookie',
12377             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12378            {'call': 'uselib',
12379             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12380            {'call': 'renameat2', 'reason': set([('dentry', 'd_inode')])},
12381            {'call': 'accept4',
12382             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12383            {'call': 'socketpair',
12384             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12385            {'call': 'getcwd',
12386             'reason': set([('dentry', 'd_inode'),
12387                            ('path', 'dentry'),
12388                            ('path', 'mnt')])},
12389            {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
12390            {'call': 'shmat',
12391             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12392            {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
12393            {'call': 'socket',
12394             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12395            {'call': 'symlinkat', 'reason': set([('dentry', 'd_inode')])},
12396            {'call': 'pipe2',
12397             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12398            {'call': 'perf_event_open',
12399             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12400            {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
12401            {'call': 'shmdt',
```

```
12402             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12403           {'call': 'quotactl',
12404             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12405           {'call': 'acct',
12406             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12407           {'call': 'open',
12408             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12409           {'call': 'unlink', 'reason': set([('dentry', 'd_inode')])},
12410           {'call': 'dup',
12411             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12412           {'call': 'setns',
12413             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12414           {'call': 'shmctl',
12415             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12416           {'call': 'swapon',
12417             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12418           {'call': 'mmap_pgoff',
12419             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
12420           {'call': 'mq_open',
12421             'reason': set([('dentry', 'd_inode'),
12422                            ('path', 'dentry'),
12423                            ('path', 'mnt')])},
12424           {'call': 'unlinkat', 'reason': set([('dentry', 'd_inode')])},
12425           {'call': 'open_by_handle_at',
12426             'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
12427  'rt_sigqueueinfo': [{'call': 'rt_sigtimedwait',
12428                       'reason': set([('siginfo', 'si_code')])},
12429                      {'call': 'kill',
12430                       'reason': set([('siginfo', 'si_code')])},
12431                      {'call': 'timer_create',
12432                       'reason': set([('siginfo', 'si_code')])},
12433                      {'call': 'tgkill',
12434                       'reason': set([('siginfo', 'si_code')])},
12435                      {'call': 'rt_tgsigqueueinfo',
12436                       'reason': set([('siginfo', 'si_code')])},
12437                      {'call': 'rt_sigreturn',
12438                       'reason': set([('siginfo', 'si_code')])},
12439                      {'call': 'tkill',
12440                       'reason': set([('siginfo', 'si_code')])}],
12441  'rt_sigreturn': [{'call': 'keyctl',
12442                    'reason': set([('mm_segment_t', 'seg'),
12443                                   ('thread_struct', 'uaccess_err')])},
12444                   {'call': 'rt_sigtimedwait',
12445                    'reason': set([('mm_segment_t', 'seg'),
12446                                   ('thread_struct', 'uaccess_err')])},
12447                   {'call': 'msgrcv',
12448                    'reason': set([('mm_segment_t', 'seg'),
12449                                   ('thread_struct', 'uaccess_err')])},
12450                   {'call': 'kill',
12451                    'reason': set([('mm_segment_t', 'seg'),
12452                                   ('thread_struct', 'uaccess_err')])},
12453                   {'call': 'sched_getaffinity',
12454                    'reason': set([('mm_segment_t', 'seg'),
12455                                   ('thread_struct', 'uaccess_err')])},
12456                   {'call': 'sched_setparam',
12457                    'reason': set([('mm_segment_t', 'seg'),
12458                                   ('thread_struct', 'uaccess_err')])},
12459                   {'call': 'ioprio_set',
12460                    'reason': set([('mm_segment_t', 'seg'),
12461                                   ('thread_struct', 'uaccess_err')])},
12462                   {'call': 'getppid',
12463                    'reason': set([('mm_segment_t', 'seg'),
12464                                   ('thread_struct', 'uaccess_err')])},
12465                   {'call': 'ioperm',
12466                    'reason': set([('mm_segment_t', 'seg'),
12467                                   ('thread_struct', 'uaccess_err')])},
```



```
12468                   {'call': 'mq_timedreceive',
12469                    'reason': set([('mm_segment_t', 'seg'),
12470                                   ('thread_struct', 'uaccess_err')])},
12471                   {'call': 'capget',
12472                    'reason': set([('mm_segment_t', 'seg'),
12473                                   ('thread_struct', 'uaccess_err')])},
12474                   {'call': 'sched_setaffinity',
12475                    'reason': set([('mm_segment_t', 'seg'),
12476                                   ('thread_struct', 'uaccess_err')])},
12477                   {'call': 'signal',
12478                    'reason': set([('mm_segment_t', 'seg'),
12479                                   ('thread_struct', 'uaccess_err')])},
12480                   {'call': 'semtimedop',
12481                    'reason': set([('mm_segment_t', 'seg'),
12482                                   ('thread_struct', 'uaccess_err')])},
12483                   {'call': 'umount',
12484                    'reason': set([('mm_segment_t', 'seg'),
12485                                   ('thread_struct', 'uaccess_err')])},
12486                   {'call': 'sched_rr_get_interval',
12487                    'reason': set([('mm_segment_t', 'seg'),
12488                                   ('thread_struct', 'uaccess_err')])},
12489                   {'call': 'rt_sigprocmask',
12490                    'reason': set([('mm_segment_t', 'seg'),
12491                                   ('thread_struct', 'uaccess_err')])},
12492                   {'call': 'setsid',
12493                    'reason': set([('mm_segment_t', 'seg'),
12494                                   ('thread_struct', 'uaccess_err')])},
12495                   {'call': 'sigaltstack',
12496                    'reason': set([('mm_segment_t', 'seg'),
12497                                   ('thread_struct', 'uaccess_err')])},
12498                   {'call': 'sched_setattr',
12499                    'reason': set([('mm_segment_t', 'seg'),
12500                                   ('thread_struct', 'uaccess_err')])},
12501                   {'call': 'migrate_pages',
12502                    'reason': set([('mm_segment_t', 'seg'),
12503                                   ('thread_struct', 'uaccess_err')])},
12504                   {'call': 'getitimer',
12505                    'reason': set([('mm_segment_t', 'seg'),
12506                                   ('thread_struct', 'uaccess_err')])},
12507                   {'call': 'setpgid',
12508                    'reason': set([('mm_segment_t', 'seg'),
12509                                   ('thread_struct', 'uaccess_err')])},
12510                   {'call': 'getsid',
12511                    'reason': set([('mm_segment_t', 'seg'),
12512                                   ('thread_struct', 'uaccess_err')])},
12513                   {'call': 'prlimit64',
12514                    'reason': set([('mm_segment_t', 'seg'),
12515                                   ('thread_struct', 'uaccess_err')])},
12516                   {'call': 'perf_event_open',
12517                    'reason': set([('mm_segment_t', 'seg'),
12518                                   ('thread_struct', 'uaccess_err')])},
12519                   {'call': 'rt_sigaction',
12520                    'reason': set([('mm_segment_t', 'seg'),
12521                                   ('thread_struct', 'uaccess_err')])},
12522                   {'call': 'getpgid',
12523                    'reason': set([('mm_segment_t', 'seg'),
12524                                   ('thread_struct', 'uaccess_err')])},
12525                   {'call': 'getpriority',
12526                    'reason': set([('mm_segment_t', 'seg'),
12527                                   ('thread_struct', 'uaccess_err')])},
12528                   {'call': 'sigaction',
12529                    'reason': set([('mm_segment_t', 'seg'),
12530                                   ('thread_struct', 'uaccess_err')])},
12531                   {'call': 'setns',
12532                    'reason': set([('mm_segment_t', 'seg'),
12533                                   ('thread_struct', 'uaccess_err')])},
```

```
12534                   {'call': 'fork',
12535                    'reason': set([('mm_segment_t', 'seg'),
12536                                   ('thread_struct', 'uaccess_err')])},
12537                   {'call': 'get_robust_list',
12538                    'reason': set([('mm_segment_t', 'seg'),
12539                                   ('thread_struct', 'uaccess_err')])},
12540                   {'call': 'mq_timedsend',
12541                    'reason': set([('mm_segment_t', 'seg'),
12542                                   ('thread_struct', 'uaccess_err')])},
12543                   {'call': 'sched_getscheduler',
12544                    'reason': set([('mm_segment_t', 'seg'),
12545                                   ('thread_struct', 'uaccess_err')])},
12546                   {'call': 'ptrace',
12547                    'reason': set([('mm_segment_t', 'seg'),
12548                                   ('thread_struct', 'uaccess_err')])},
12549                   {'call': 'sched_getattr',
12550                    'reason': set([('mm_segment_t', 'seg'),
12551                                   ('thread_struct', 'uaccess_err')])},
12552                   {'call': 'getrusage',
12553                    'reason': set([('mm_segment_t', 'seg'),
12554                                   ('thread_struct', 'uaccess_err')])},
12555                   {'call': 'sched_setscheduler',
12556                    'reason': set([('mm_segment_t', 'seg'),
12557                                   ('thread_struct', 'uaccess_err')])},
12558                   {'call': 'setitimer',
12559                    'reason': set([('mm_segment_t', 'seg'),
12560                                   ('thread_struct', 'uaccess_err')])},
12561                   {'call': 'ioprio_get',
12562                    'reason': set([('mm_segment_t', 'seg'),
12563                                   ('thread_struct', 'uaccess_err')])},
12564                   {'call': 'vfork',
12565                    'reason': set([('mm_segment_t', 'seg'),
12566                                   ('thread_struct', 'uaccess_err')])},
12567                   {'call': 'prctl',
12568                    'reason': set([('mm_segment_t', 'seg'),
12569                                   ('thread_struct', 'uaccess_err')])},
12570                   {'call': 'move_pages',
12571                    'reason': set([('mm_segment_t', 'seg'),
12572                                   ('thread_struct', 'uaccess_err')])},
12573                   {'call': 'setpriority',
12574                    'reason': set([('mm_segment_t', 'seg'),
12575                                   ('thread_struct', 'uaccess_err')])},
12576                   {'call': 'clone',
12577                    'reason': set([('mm_segment_t', 'seg'),
12578                                   ('thread_struct', 'uaccess_err')])},
12579                   {'call': 'sched_getparam',
12580                    'reason': set([('mm_segment_t', 'seg'),
12581                                   ('thread_struct', 'uaccess_err')])}],
12582  'rt_sigtimedwait': [{'call': 'keyctl',
12583                       'reason': set([('mm_segment_t', 'seg'),
12584                                      ('task_struct', 'timer_slack_ns')])},
12585                      {'call': 'msgrcv',
12586                       'reason': set([('mm_segment_t', 'seg'),
12587                                      ('task_struct', 'timer_slack_ns')])},
12588                      {'call': 'kill',
12589                       'reason': set([('mm_segment_t', 'seg'),
12590                                      ('siginfo', 'si_code'),
12591                                      ('siginfo', 'si_signo'),
12592                                      ('task_struct', 'timer_slack_ns')])},
12593                      {'call': 'sched_getaffinity',
12594                       'reason': set([('mm_segment_t', 'seg'),
12595                                      ('task_struct', 'timer_slack_ns')])},
12596                      {'call': 'sched_setparam',
12597                       'reason': set([('mm_segment_t', 'seg'),
12598                                      ('task_struct', 'timer_slack_ns')])},
12599                      {'call': 'ioprio_set',
```



```
12600                       'reason': set([('mm_segment_t', 'seg'),
12601                                      ('task_struct', 'timer_slack_ns')])},
12602                     {'call': 'getppid',
12603                      'reason': set([('mm_segment_t', 'seg'),
12604                                     ('task_struct', 'timer_slack_ns')])},
12605                     {'call': 'ioperm',
12606                      'reason': set([('mm_segment_t', 'seg')])},
12607                     {'call': 'mq_timedreceive',
12608                      'reason': set([('mm_segment_t', 'seg'),
12609                                     ('task_struct', 'timer_slack_ns')])},
12610                     {'call': 'capget',
12611                      'reason': set([('mm_segment_t', 'seg'),
12612                                     ('task_struct', 'timer_slack_ns')])},
12613                     {'call': 'sched_setaffinity',
12614                      'reason': set([('mm_segment_t', 'seg'),
12615                                     ('task_struct', 'timer_slack_ns')])},
12616                     {'call': 'signal',
12617                      'reason': set([('mm_segment_t', 'seg'),
12618                                     ('task_struct', 'timer_slack_ns')])},
12619                     {'call': 'semtimedop',
12620                      'reason': set([('mm_segment_t', 'seg'),
12621                                     ('task_struct', 'timer_slack_ns')])},
12622                     {'call': 'umount',
12623                      'reason': set([('mm_segment_t', 'seg'),
12624                                     ('task_struct', 'timer_slack_ns')])},
12625                     {'call': 'timer_create',
12626                      'reason': set([('siginfo', 'si_code'),
12627                                     ('siginfo', 'si_signo')])},
12628                     {'call': 'sched_rr_get_interval',
12629                      'reason': set([('mm_segment_t', 'seg'),
12630                                     ('task_struct', 'timer_slack_ns')])},
12631                     {'call': 'rt_sigqueueinfo',
12632                      'reason': set([('siginfo', 'si_code'),
12633                                     ('siginfo', 'si_signo')])},
12634                     {'call': 'tgkill',
12635                      'reason': set([('siginfo', 'si_code'),
12636                                     ('siginfo', 'si_signo')])},
12637                     {'call': 'rt_sigprocmask',
12638                      'reason': set([('mm_segment_t', 'seg'),
12639                                     ('task_struct', 'timer_slack_ns')])},
12640                     {'call': 'setsid',
12641                      'reason': set([('mm_segment_t', 'seg'),
12642                                     ('task_struct', 'timer_slack_ns')])},
12643                     {'call': 'sigaltstack',
12644                      'reason': set([('mm_segment_t', 'seg'),
12645                                     ('task_struct', 'timer_slack_ns')])},
12646                     {'call': 'sched_setattr',
12647                      'reason': set([('mm_segment_t', 'seg'),
12648                                     ('task_struct', 'timer_slack_ns')])},
12649                     {'call': 'migrate_pages',
12650                      'reason': set([('mm_segment_t', 'seg'),
12651                                     ('task_struct', 'timer_slack_ns')])},
12652                     {'call': 'getitimer',
12653                      'reason': set([('mm_segment_t', 'seg'),
12654                                     ('task_struct', 'timer_slack_ns')])},
12655                     {'call': 'setpgid',
12656                      'reason': set([('mm_segment_t', 'seg'),
12657                                     ('task_struct', 'timer_slack_ns')])},
12658                     {'call': 'getsid',
12659                      'reason': set([('mm_segment_t', 'seg'),
12660                                     ('task_struct', 'timer_slack_ns')])},
12661                     {'call': 'prlimit64',
12662                      'reason': set([('mm_segment_t', 'seg'),
12663                                     ('task_struct', 'timer_slack_ns')])},
12664                     {'call': 'perf_event_open',
12665                      'reason': set([('mm_segment_t', 'seg'),
```

```
                                    ('task_struct', 'timer_slack_ns')])},
                        {'call': 'rt_sigaction',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'getpgid',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'getpriority',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'sigaction',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'rt_tgsigqueueinfo',
                         'reason': set([('siginfo', 'si_code'),
                                        ('siginfo', 'si_signo')])},
                        {'call': 'setns',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'fork',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'get_robust_list',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'mq_timedsend',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'sched_getscheduler',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'ptrace',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'sched_getattr',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'getrusage',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'sched_setscheduler',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'setitimer',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'ioprio_get',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'vfork',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'prctl',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'move_pages',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'rt_sigreturn',
                         'reason': set([('siginfo', 'si_code'),
                                        ('siginfo', 'si_signo')])},
                        {'call': 'tkill',
                         'reason': set([('siginfo', 'si_code'),
                                        ('siginfo', 'si_signo')])},
                        {'call': 'setpriority',
                         'reason': set([('mm_segment_t', 'seg'),
```



```
                                    ('task_struct', 'timer_slack_ns')])},
                        {'call': 'clone',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])},
                        {'call': 'sched_getparam',
                         'reason': set([('mm_segment_t', 'seg'),
                                        ('task_struct', 'timer_slack_ns')])}],
 'rt_tgsigqueueinfo': [{'call': 'rt_sigtimedwait',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'kill',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'timer_create',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'rt_sigqueueinfo',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'tgkill',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'rt_sigreturn',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'tkill',
                        'reason': set([('siginfo', 'si_code')])}],
 'sched_getattr': [{'call': 'keyctl',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'msgrcv',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'kill',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_setparam',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'ioprio_set',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'getppid',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'ioperm',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'capget',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_setaffinity',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
```

```
12798                                 ('task_struct', 'sched_reset_on_fork')])},
12799                     {'call': 'signal',
12800                      'reason': set([('mm_segment_t', 'seg'),
12801                                     ('task_struct', 'policy'),
12802                                     ('task_struct', 'sched_reset_on_fork')])},
12803                     {'call': 'semtimedop',
12804                      'reason': set([('mm_segment_t', 'seg'),
12805                                     ('task_struct', 'policy'),
12806                                     ('task_struct', 'sched_reset_on_fork')])},
12807                     {'call': 'umount',
12808                      'reason': set([('mm_segment_t', 'seg'),
12809                                     ('task_struct', 'policy'),
12810                                     ('task_struct', 'sched_reset_on_fork')])},
12811                     {'call': 'sched_rr_get_interval',
12812                      'reason': set([('mm_segment_t', 'seg'),
12813                                     ('task_struct', 'policy'),
12814                                     ('task_struct', 'sched_reset_on_fork')])},
12815                     {'call': 'rt_sigprocmask',
12816                      'reason': set([('mm_segment_t', 'seg'),
12817                                     ('task_struct', 'policy'),
12818                                     ('task_struct', 'sched_reset_on_fork')])},
12819                     {'call': 'setsid',
12820                      'reason': set([('mm_segment_t', 'seg'),
12821                                     ('task_struct', 'policy'),
12822                                     ('task_struct', 'sched_reset_on_fork')])},
12823                     {'call': 'sigaltstack',
12824                      'reason': set([('mm_segment_t', 'seg'),
12825                                     ('task_struct', 'policy'),
12826                                     ('task_struct', 'sched_reset_on_fork')])},
12827                     {'call': 'sched_setattr',
12828                      'reason': set([('mm_segment_t', 'seg'),
12829                                     ('sched_attr', 'size'),
12830                                     ('task_struct', 'policy'),
12831                                     ('task_struct', 'sched_reset_on_fork')])},
12832                     {'call': 'migrate_pages',
12833                      'reason': set([('mm_segment_t', 'seg'),
12834                                     ('task_struct', 'policy'),
12835                                     ('task_struct', 'sched_reset_on_fork')])},
12836                     {'call': 'getitimer',
12837                      'reason': set([('mm_segment_t', 'seg'),
12838                                     ('task_struct', 'policy'),
12839                                     ('task_struct', 'sched_reset_on_fork')])},
12840                     {'call': 'setpgid',
12841                      'reason': set([('mm_segment_t', 'seg'),
12842                                     ('task_struct', 'policy'),
12843                                     ('task_struct', 'sched_reset_on_fork')])},
12844                     {'call': 'getsid',
12845                      'reason': set([('mm_segment_t', 'seg'),
12846                                     ('task_struct', 'policy'),
12847                                     ('task_struct', 'sched_reset_on_fork')])},
12848                     {'call': 'prlimit64',
12849                      'reason': set([('mm_segment_t', 'seg'),
12850                                     ('task_struct', 'policy'),
12851                                     ('task_struct', 'sched_reset_on_fork')])},
12852                     {'call': 'perf_event_open',
12853                      'reason': set([('mm_segment_t', 'seg'),
12854                                     ('task_struct', 'policy'),
12855                                     ('task_struct', 'sched_reset_on_fork')])},
12856                     {'call': 'rt_sigaction',
12857                      'reason': set([('mm_segment_t', 'seg'),
12858                                     ('task_struct', 'policy'),
12859                                     ('task_struct', 'sched_reset_on_fork')])},
12860                     {'call': 'getpgid',
12861                      'reason': set([('mm_segment_t', 'seg'),
12862                                     ('task_struct', 'policy'),
12863                                     ('task_struct', 'sched_reset_on_fork')])},
```



```
12864                     {'call': 'getpriority',
12865                      'reason': set([('mm_segment_t', 'seg'),
12866                                     ('task_struct', 'policy'),
12867                                     ('task_struct', 'sched_reset_on_fork')])},
12868                     {'call': 'sigaction',
12869                      'reason': set([('mm_segment_t', 'seg'),
12870                                     ('task_struct', 'policy'),
12871                                     ('task_struct', 'sched_reset_on_fork')])},
12872                     {'call': 'setns',
12873                      'reason': set([('mm_segment_t', 'seg'),
12874                                     ('task_struct', 'policy'),
12875                                     ('task_struct', 'sched_reset_on_fork')])},
12876                     {'call': 'fork',
12877                      'reason': set([('mm_segment_t', 'seg'),
12878                                     ('task_struct', 'policy'),
12879                                     ('task_struct', 'sched_reset_on_fork')])},
12880                     {'call': 'get_robust_list',
12881                      'reason': set([('mm_segment_t', 'seg'),
12882                                     ('task_struct', 'policy'),
12883                                     ('task_struct', 'sched_reset_on_fork')])},
12884                     {'call': 'mq_timedsend',
12885                      'reason': set([('mm_segment_t', 'seg'),
12886                                     ('task_struct', 'policy'),
12887                                     ('task_struct', 'sched_reset_on_fork')])},
12888                     {'call': 'sched_getscheduler',
12889                      'reason': set([('mm_segment_t', 'seg'),
12890                                     ('task_struct', 'policy'),
12891                                     ('task_struct', 'sched_reset_on_fork')])},
12892                     {'call': 'ptrace',
12893                      'reason': set([('mm_segment_t', 'seg'),
12894                                     ('task_struct', 'policy'),
12895                                     ('task_struct', 'sched_reset_on_fork')])},
12896                     {'call': 'getrusage',
12897                      'reason': set([('mm_segment_t', 'seg'),
12898                                     ('task_struct', 'policy'),
12899                                     ('task_struct', 'sched_reset_on_fork')])},
12900                     {'call': 'sched_setscheduler',
12901                      'reason': set([('mm_segment_t', 'seg'),
12902                                     ('task_struct', 'policy'),
12903                                     ('task_struct', 'sched_reset_on_fork')])},
12904                     {'call': 'setitimer',
12905                      'reason': set([('mm_segment_t', 'seg'),
12906                                     ('task_struct', 'policy'),
12907                                     ('task_struct', 'sched_reset_on_fork')])},
12908                     {'call': 'ioprio_get',
12909                      'reason': set([('mm_segment_t', 'seg'),
12910                                     ('task_struct', 'policy'),
12911                                     ('task_struct', 'sched_reset_on_fork')])},
12912                     {'call': 'vfork',
12913                      'reason': set([('mm_segment_t', 'seg'),
12914                                     ('task_struct', 'policy'),
12915                                     ('task_struct', 'sched_reset_on_fork')])},
12916                     {'call': 'prctl',
12917                      'reason': set([('mm_segment_t', 'seg'),
12918                                     ('task_struct', 'policy'),
12919                                     ('task_struct', 'sched_reset_on_fork')])},
12920                     {'call': 'move_pages',
12921                      'reason': set([('mm_segment_t', 'seg'),
12922                                     ('task_struct', 'policy'),
12923                                     ('task_struct', 'sched_reset_on_fork')])},
12924                     {'call': 'setpriority',
12925                      'reason': set([('mm_segment_t', 'seg'),
12926                                     ('task_struct', 'policy'),
12927                                     ('task_struct', 'sched_reset_on_fork')])},
12928                     {'call': 'clone',
12929                      'reason': set([('mm_segment_t', 'seg'),
```

```
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_getparam',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])}],
 'sched_getparam': [{'call': 'keyctl',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigtimedwait',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'msgrcv',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'kill',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getaffinity',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setparam',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'ioprio_set',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getppid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'mq_timedreceive',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'capget',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setaffinity',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'signal',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'semtimedop',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'umount',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_rr_get_interval',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigprocmask',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setsid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sigaltstack',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setattr',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'migrate_pages',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getitimer',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setpgid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getsid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'prlimit64',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'perf_event_open',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigaction',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getpgid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getpriority',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sigaction',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setns',
                     'reason': set([('task_struct', 'policy')])},
```



```
                    {'call': 'fork',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'get_robust_list',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'ptrace',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getattr',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getrusage',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setitimer',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'ioprio_get',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'vfork',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'prctl',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'move_pages',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setpriority',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'clone',
                     'reason': set([('task_struct', 'policy')])}],
 'sched_getscheduler': [{'call': 'keyctl',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'rt_sigtimedwait',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'msgrcv',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'kill',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_getaffinity',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_setparam',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'ioprio_set',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'getppid',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'mq_timedreceive',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'capget',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_setaffinity',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'signal',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
```

```
                        {'call': 'semtimedop',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'umount',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_rr_get_interval',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'rt_sigprocmask',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'setsid',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sigaltstack',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_setattr',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'migrate_pages',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'getitimer',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'setpgid',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'getsid',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'prlimit64',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'perf_event_open',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'rt_sigaction',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'getpgid',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'getpriority',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sigaction',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'setns',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'fork',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'get_robust_list',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'mq_timedsend',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'ptrace',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
```



```
                        {'call': 'sched_getattr',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'getrusage',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_setscheduler',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'setitimer',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'ioprio_get',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'vfork',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'prctl',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'move_pages',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'setpriority',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'clone',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])},
                        {'call': 'sched_getparam',
                         'reason': set([('task_struct',
                                         'sched_reset_on_fork')])}],
 'sched_setaffinity': [{'call': 'keyctl',
                        'reason': set([('cred', 'user_ns'),
                                       ('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'rt_sigtimedwait',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'setfsuid',
                        'reason': set([('cred', 'user_ns')])},
                       {'call': 'msgrcv',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'kill',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'getresuid16',
                        'reason': set([('cred', 'user_ns')])},
                       {'call': 'getresgid',
                        'reason': set([('cred', 'user_ns')])},
                       {'call': 'sched_getaffinity',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'sched_setparam',
                        'reason': set([('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'setgid',
                        'reason': set([('cred', 'user_ns')])},
                       {'call': 'ioprio_set',
                        'reason': set([('cred', 'user_ns'),
                                       ('task_struct', 'flags'),
                                       ('task_struct', 'real_cred')])},
                       {'call': 'capset',
                        'reason': set([('cred', 'user_ns')])},
```

```
13194                         {'call': 'getppid',
13195                          'reason': set([('task_struct', 'flags'),
13196                                         ('task_struct', 'real_cred')])},
13197                         {'call': 'mq_timedreceive',
13198                          'reason': set([('task_struct', 'flags'),
13199                                         ('task_struct', 'real_cred')])},
13200                         {'call': 'getresgid16',
13201                          'reason': set([('cred', 'user_ns')])},
13202                         {'call': 'capget',
13203                          'reason': set([('task_struct', 'flags'),
13204                                         ('task_struct', 'real_cred')])},
13205                         {'call': 'setfsgid',
13206                          'reason': set([('cred', 'user_ns')])},
13207                         {'call': 'unshare',
13208                          'reason': set([('cred', 'user_ns')])},
13209                         {'call': 'signal',
13210                          'reason': set([('task_struct', 'flags'),
13211                                         ('task_struct', 'real_cred')])},
13212                         {'call': 'setreuid',
13213                          'reason': set([('cred', 'user_ns'),
13214                                         ('task_struct', 'flags')])},
13215                         {'call': 'semtimedop',
13216                          'reason': set([('task_struct', 'flags'),
13217                                         ('task_struct', 'real_cred')])},
13218                         {'call': 'umount',
13219                          'reason': set([('task_struct', 'flags'),
13220                                         ('task_struct', 'real_cred')])},
13221                         {'call': 'sched_rr_get_interval',
13222                          'reason': set([('task_struct', 'flags'),
13223                                         ('task_struct', 'real_cred')])},
13224                         {'call': 'epoll_create1',
13225                          'reason': set([('cred', 'user_ns')])},
13226                         {'call': 'getresuid',
13227                          'reason': set([('cred', 'user_ns')])},
13228                         {'call': 'rt_sigprocmask',
13229                          'reason': set([('task_struct', 'flags'),
13230                                         ('task_struct', 'real_cred')])},
13231                         {'call': 'setsid',
13232                          'reason': set([('task_struct', 'flags'),
13233                                         ('task_struct', 'real_cred')])},
13234                         {'call': 'sigaltstack',
13235                          'reason': set([('task_struct', 'flags'),
13236                                         ('task_struct', 'real_cred')])},
13237                         {'call': 'sched_setattr',
13238                          'reason': set([('task_struct', 'flags'),
13239                                         ('task_struct', 'real_cred')])},
13240                         {'call': 'migrate_pages',
13241                          'reason': set([('cred', 'user_ns'),
13242                                         ('task_struct', 'flags'),
13243                                         ('task_struct', 'real_cred')])},
13244                         {'call': 'getitimer',
13245                          'reason': set([('task_struct', 'flags'),
13246                                         ('task_struct', 'real_cred')])},
13247                         {'call': 'setpgid',
13248                          'reason': set([('task_struct', 'flags'),
13249                                         ('task_struct', 'real_cred')])},
13250                         {'call': 'setresgid',
13251                          'reason': set([('cred', 'user_ns')])},
13252                         {'call': 'setregid',
13253                          'reason': set([('cred', 'user_ns')])},
13254                         {'call': 'getsid',
13255                          'reason': set([('task_struct', 'flags'),
13256                                         ('task_struct', 'real_cred')])},
13257                         {'call': 'prlimit64',
13258                          'reason': set([('cred', 'user_ns'),
13259                                         ('task_struct', 'flags'),
```



```
13260                                         ('task_struct', 'real_cred')])},
13261                         {'call': 'perf_event_open',
13262                          'reason': set([('task_struct', 'flags'),
13263                                         ('task_struct', 'real_cred')])},
13264                         {'call': 'getgroups16',
13265                          'reason': set([('cred', 'user_ns')])},
13266                         {'call': 'rt_sigaction',
13267                          'reason': set([('task_struct', 'flags'),
13268                                         ('task_struct', 'real_cred')])},
13269                         {'call': 'getpgid',
13270                          'reason': set([('task_struct', 'flags'),
13271                                         ('task_struct', 'real_cred')])},
13272                         {'call': 'getpriority',
13273                          'reason': set([('cred', 'user_ns'),
13274                                         ('task_struct', 'flags'),
13275                                         ('task_struct', 'real_cred')])},
13276                         {'call': 'sigaction',
13277                          'reason': set([('task_struct', 'flags'),
13278                                         ('task_struct', 'real_cred')])},
13279                         {'call': 'faccessat',
13280                          'reason': set([('cred', 'user_ns')])},
13281                         {'call': 'setns',
13282                          'reason': set([('task_struct', 'flags'),
13283                                         ('task_struct', 'real_cred')])},
13284                         {'call': 'fork',
13285                          'reason': set([('task_struct', 'flags'),
13286                                         ('task_struct', 'real_cred')])},
13287                         {'call': 'get_robust_list',
13288                          'reason': set([('task_struct', 'flags'),
13289                                         ('task_struct', 'real_cred')])},
13290                         {'call': 'mq_timedsend',
13291                          'reason': set([('task_struct', 'flags'),
13292                                         ('task_struct', 'real_cred')])},
13293                         {'call': 'sched_getscheduler',
13294                          'reason': set([('task_struct', 'flags'),
13295                                         ('task_struct', 'real_cred')])},
13296                         {'call': 'ptrace',
13297                          'reason': set([('task_struct', 'flags'),
13298                                         ('task_struct', 'real_cred')])},
13299                         {'call': 'sched_getattr',
13300                          'reason': set([('task_struct', 'flags'),
13301                                         ('task_struct', 'real_cred')])},
13302                         {'call': 'getrusage',
13303                          'reason': set([('task_struct', 'flags'),
13304                                         ('task_struct', 'real_cred')])},
13305                         {'call': 'sched_setscheduler',
13306                          'reason': set([('task_struct', 'flags'),
13307                                         ('task_struct', 'real_cred')])},
13308                         {'call': 'setresuid',
13309                          'reason': set([('cred', 'user_ns'),
13310                                         ('task_struct', 'flags')])},
13311                         {'call': 'setitimer',
13312                          'reason': set([('task_struct', 'flags'),
13313                                         ('task_struct', 'real_cred')])},
13314                         {'call': 'ioprio_get',
13315                          'reason': set([('cred', 'user_ns'),
13316                                         ('task_struct', 'flags'),
13317                                         ('task_struct', 'real_cred')])},
13318                         {'call': 'vfork',
13319                          'reason': set([('task_struct', 'flags'),
13320                                         ('task_struct', 'real_cred')])},
13321                         {'call': 'setuid',
13322                          'reason': set([('cred', 'user_ns'),
13323                                         ('task_struct', 'flags')])},
13324                         {'call': 'prctl',
13325                          'reason': set([('task_struct', 'flags'),
```

```
                                        ('task_struct', 'real_cred')])},
                        {'call': 'move_pages',
                         'reason': set([('task_struct', 'flags'),
                                        ('task_struct', 'real_cred')])},
                        {'call': 'getgroups',
                         'reason': set([('cred', 'user_ns')])},
                        {'call': 'setpriority',
                         'reason': set([('cred', 'user_ns'),
                                        ('task_struct', 'flags'),
                                        ('task_struct', 'real_cred')])},
                        {'call': 'clone',
                         'reason': set([('task_struct', 'flags'),
                                        ('task_struct', 'real_cred')])},
                        {'call': 'sched_getparam',
                         'reason': set([('task_struct', 'flags'),
                                        ('task_struct', 'real_cred')])}],
 'sched_setattr': [{'call': 'keyctl',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'msgrcv',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_setparam',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ioprio_set',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getppid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ioperm',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'capget',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_setaffinity',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'signal',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'semtimedop',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'umount',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_rr_get_interval',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'rt_sigprocmask',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setsid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sigaltstack',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'migrate_pages',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getitimer',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setpgid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getsid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'prlimit64',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'perf_event_open',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'rt_sigaction',
```



```
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getpgid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getpriority',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sigaction',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setns',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'get_robust_list',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'mq_timedsend',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_getscheduler',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ptrace',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_getattr',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('sched_attr', 'sched_policy')])},
                   {'call': 'getrusage',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_setscheduler',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setitimer',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ioprio_get',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'vfork',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'prctl',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'move_pages',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setpriority',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'clone',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_getparam',
                    'reason': set([('mm_segment_t', 'seg')])}],
 'select': [{'call': 'keyctl',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'personality'),
                            ('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'msgrcv',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'mq_unlink',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
            {'call': 'swapoff',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'fchmod',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'memfd_create',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
```

```
13458             {'call': 'ioprio_set',
13459              'reason': set([('task_struct', 'personality')])},
13460             {'call': 'personality',
13461              'reason': set([('task_struct', 'personality')])},
13462             {'call': 'dup3', 'reason': set([('files_struct', 'fdt')])},
13463             {'call': 'readlinkat',
13464              'reason': set([('timespec', 'tv_nsec'),
13465                             ('timespec', 'tv_sec')])},
13466             {'call': 'io_getevents',
13467              'reason': set([('timespec', 'tv_nsec'),
13468                             ('timespec', 'tv_sec')])},
13469             {'call': 'getppid',
13470              'reason': set([('task_struct', 'personality')])},
13471             {'call': 'fchown',
13472              'reason': set([('timespec', 'tv_nsec'),
13473                             ('timespec', 'tv_sec')])},
13474             {'call': 'mq_timedreceive',
13475              'reason': set([('task_struct', 'personality'),
13476                             ('timespec', 'tv_nsec'),
13477                             ('timespec', 'tv_sec')])},
13478             {'call': 'capget',
13479              'reason': set([('task_struct', 'personality')])},
13480             {'call': 'utime',
13481              'reason': set([('timespec', 'tv_nsec'),
13482                             ('timespec', 'tv_sec')])},
13483             {'call': 'sched_setaffinity',
13484              'reason': set([('task_struct', 'personality')])},
13485             {'call': 'unshare', 'reason': set([('files_struct', 'fdt')])},
13486             {'call': 'signal',
13487              'reason': set([('task_struct', 'personality')])},
13488             {'call': 'semtimedop',
13489              'reason': set([('task_struct', 'personality'),
13490                             ('timespec', 'tv_nsec'),
13491                             ('timespec', 'tv_sec')])},
13492             {'call': 'umount',
13493              'reason': set([('task_struct', 'personality')])},
13494             {'call': 'settimeofday',
13495              'reason': set([('timespec', 'tv_nsec'),
13496                             ('timespec', 'tv_sec')])},
13497             {'call': 'sched_rr_get_interval',
13498              'reason': set([('task_struct', 'personality'),
13499                             ('timespec', 'tv_nsec'),
13500                             ('timespec', 'tv_sec')])},
13501             {'call': 'timerfd_gettime',
13502              'reason': set([('timespec', 'tv_nsec'),
13503                             ('timespec', 'tv_sec')])},
13504             {'call': 'pselect6',
13505              'reason': set([('timespec', 'tv_nsec'),
13506                             ('timespec', 'tv_sec')])},
13507             {'call': 'uselib',
13508              'reason': set([('timespec', 'tv_nsec'),
13509                             ('timespec', 'tv_sec')])},
13510             {'call': 'rt_sigprocmask',
13511              'reason': set([('task_struct', 'personality')])},
13512             {'call': 'setsid',
13513              'reason': set([('task_struct', 'personality')])},
13514             {'call': 'sigaltstack',
13515              'reason': set([('task_struct', 'personality')])},
13516             {'call': 'sched_setattr',
13517              'reason': set([('task_struct', 'personality')])},
13518             {'call': 'migrate_pages',
13519              'reason': set([('task_struct', 'personality')])},
13520             {'call': 'getitimer',
13521              'reason': set([('task_struct', 'personality')])},
13522             {'call': 'fchmodat',
13523              'reason': set([('timespec', 'tv_nsec'),
```



```
13524                             ('timespec', 'tv_sec')])},
13525             {'call': 'setpgid',
13526              'reason': set([('task_struct', 'personality')])},
13527             {'call': 'inotify_add_watch',
13528              'reason': set([('timespec', 'tv_nsec'),
13529                             ('timespec', 'tv_sec')])},
13530             {'call': 'timer_settime',
13531              'reason': set([('timespec', 'tv_nsec'),
13532                             ('timespec', 'tv_sec')])},
13533             {'call': 'ftruncate',
13534              'reason': set([('timespec', 'tv_nsec'),
13535                             ('timespec', 'tv_sec')])},
13536             {'call': 'timer_gettime',
13537              'reason': set([('timespec', 'tv_nsec'),
13538                             ('timespec', 'tv_sec')])},
13539             {'call': 'getsid',
13540              'reason': set([('task_struct', 'personality')])},
13541             {'call': 'ioctl',
13542              'reason': set([('timespec', 'tv_nsec'),
13543                             ('timespec', 'tv_sec')])},
13544             {'call': 'prlimit64',
13545              'reason': set([('task_struct', 'personality')])},
13546             {'call': 'perf_event_open',
13547              'reason': set([('task_struct', 'personality')])},
13548             {'call': 'linkat',
13549              'reason': set([('timespec', 'tv_nsec'),
13550                             ('timespec', 'tv_sec')])},
13551             {'call': 'stime',
13552              'reason': set([('timespec', 'tv_nsec'),
13553                             ('timespec', 'tv_sec')])},
13554             {'call': 'rt_sigaction',
13555              'reason': set([('task_struct', 'personality')])},
13556             {'call': 'futimesat',
13557              'reason': set([('timespec', 'tv_nsec'),
13558                             ('timespec', 'tv_sec')])},
13559             {'call': 'getpgid',
13560              'reason': set([('task_struct', 'personality')])},
13561             {'call': 'poll',
13562              'reason': set([('timespec', 'tv_nsec'),
13563                             ('timespec', 'tv_sec')])},
13564             {'call': 'unlink',
13565              'reason': set([('timespec', 'tv_nsec'),
13566                             ('timespec', 'tv_sec')])},
13567             {'call': 'getpriority',
13568              'reason': set([('task_struct', 'personality')])},
13569             {'call': 'sigaction',
13570              'reason': set([('task_struct', 'personality')])},
13571             {'call': 'nanosleep',
13572              'reason': set([('timespec', 'tv_nsec'),
13573                             ('timespec', 'tv_sec')])},
13574             {'call': 'mq_getsetattr',
13575              'reason': set([('timespec', 'tv_nsec'),
13576                             ('timespec', 'tv_sec')])},
13577             {'call': 'faccessat',
13578              'reason': set([('timespec', 'tv_nsec'),
13579                             ('timespec', 'tv_sec')])},
13580             {'call': 'setns',
13581              'reason': set([('task_struct', 'personality')])},
13582             {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
13583             {'call': 'get_robust_list',
13584              'reason': set([('task_struct', 'personality')])},
13585             {'call': 'mq_timedsend',
13586              'reason': set([('task_struct', 'personality'),
13587                             ('timespec', 'tv_nsec'),
13588                             ('timespec', 'tv_sec')])},
13589             {'call': 'sched_getscheduler',
```

```
13590              'reason': set([('task_struct', 'personality')])},
13591             {'call': 'ptrace',
13592              'reason': set([('task_struct', 'personality')])},
13593             {'call': 'swapon',
13594              'reason': set([('timespec', 'tv_nsec'),
13595                             ('timespec', 'tv_sec')])},
13596             {'call': 'epoll_wait',
13597              'reason': set([('timespec', 'tv_nsec'),
13598                             ('timespec', 'tv_sec')])},
13599             {'call': 'sched_getattr',
13600              'reason': set([('task_struct', 'personality')])},
13601             {'call': 'fchownat',
13602              'reason': set([('timespec', 'tv_nsec'),
13603                             ('timespec', 'tv_sec')])},
13604             {'call': 'getrusage',
13605              'reason': set([('task_struct', 'personality')])},
13606             {'call': 'fstat',
13607              'reason': set([('timespec', 'tv_nsec'),
13608                             ('timespec', 'tv_sec')])},
13609             {'call': 'timerfd_settime',
13610              'reason': set([('timespec', 'tv_nsec'),
13611                             ('timespec', 'tv_sec')])},
13612             {'call': 'sched_setscheduler',
13613              'reason': set([('task_struct', 'personality')])},
13614             {'call': 'setitimer',
13615              'reason': set([('task_struct', 'personality')])},
13616             {'call': 'ioprio_get',
13617              'reason': set([('task_struct', 'personality')])},
13618             {'call': 'vfork',
13619              'reason': set([('task_struct', 'personality')])},
13620             {'call': 'prctl',
13621              'reason': set([('task_struct', 'personality')])},
13622             {'call': 'move_pages',
13623              'reason': set([('task_struct', 'personality')])},
13624             {'call': 'dup2', 'reason': set([('files_struct', 'fdt')])},
13625             {'call': 'setpriority',
13626              'reason': set([('task_struct', 'personality')])},
13627             {'call': 'mq_notify',
13628              'reason': set([('timespec', 'tv_nsec'),
13629                             ('timespec', 'tv_sec')])},
13630             {'call': 'sendfile',
13631              'reason': set([('timespec', 'tv_nsec'),
13632                             ('timespec', 'tv_sec')])},
13633             {'call': 'newfstat',
13634              'reason': set([('timespec', 'tv_nsec'),
13635                             ('timespec', 'tv_sec')])},
13636             {'call': 'clone',
13637              'reason': set([('task_struct', 'personality')])},
13638             {'call': 'clock_nanosleep',
13639              'reason': set([('timespec', 'tv_nsec'),
13640                             ('timespec', 'tv_sec')])},
13641             {'call': 'unlinkat',
13642              'reason': set([('timespec', 'tv_nsec'),
13643                             ('timespec', 'tv_sec')])},
13644             {'call': 'sched_getparam',
13645              'reason': set([('task_struct', 'personality')])},
13646             {'call': 'futex',
13647              'reason': set([('timespec', 'tv_nsec'),
13648                             ('timespec', 'tv_sec')])},
13649             {'call': 'recvmmsg',
13650              'reason': set([('timespec', 'tv_nsec'),
13651                             ('timespec', 'tv_sec')])},
13652             {'call': 'sendfile64',
13653              'reason': set([('timespec', 'tv_nsec'),
13654                             ('timespec', 'tv_sec')])},
13655             {'call': 'ppoll',
```



```
13656              'reason': set([('timespec', 'tv_nsec'),
13657                             ('timespec', 'tv_sec')])}],
13658  'semctl': [{'call': 'semtimedop',
13659              'reason': set([('sem_array', 'sem_nsems')])}],
13660  'semtimedop': [{'call': 'syncfs', 'reason': set([('list_head', 'prev')])},
13661                 {'call': 'keyctl', 'reason': set([('list_head', 'prev')])},
13662                 {'call': 'rt_sigtimedwait',
13663                  'reason': set([('list_head', 'prev'),
13664                                 ('timespec', 'tv_nsec'),
13665                                 ('timespec', 'tv_sec')])},
13666                 {'call': 'vmsplice', 'reason': set([('list_head', 'prev')])},
13667                 {'call': 'msgrcv',
13668                  'reason': set([('kern_ipc_perm', 'deleted'),
13669                                 ('list_head', 'prev')])},
13670                 {'call': 'eventfd2', 'reason': set([('list_head', 'prev')])},
13671                 {'call': 'mq_unlink',
13672                  'reason': set([('list_head', 'prev'),
13673                                 ('timespec', 'tv_nsec'),
13674                                 ('timespec', 'tv_sec')])},
13675                 {'call': 'kill', 'reason': set([('list_head', 'prev')])},
13676                 {'call': 'swapoff',
13677                  'reason': set([('list_head', 'prev'),
13678                                 ('timespec', 'tv_nsec'),
13679                                 ('timespec', 'tv_sec')])},
13680                 {'call': 'readahead', 'reason': set([('list_head', 'prev')])},
13681                 {'call': 'timer_delete',
13682                  'reason': set([('list_head', 'prev')])},
13683                 {'call': 'sched_getaffinity',
13684                  'reason': set([('list_head', 'prev')])},
13685                 {'call': 'sched_setparam',
13686                  'reason': set([('list_head', 'prev')])},
13687                 {'call': 'fchmod',
13688                  'reason': set([('list_head', 'prev'),
13689                                 ('timespec', 'tv_nsec'),
13690                                 ('timespec', 'tv_sec')])},
13691                 {'call': 'setgid', 'reason': set([('list_head', 'prev')])},
13692                 {'call': 'pivot_root',
13693                  'reason': set([('list_head', 'prev')])},
13694                 {'call': 'memfd_create',
13695                  'reason': set([('list_head', 'prev'),
13696                                 ('timespec', 'tv_nsec'),
13697                                 ('timespec', 'tv_sec')])},
13698                 {'call': 'ioprio_set',
13699                  'reason': set([('list_head', 'prev')])},
13700                 {'call': 'delete_module',
13701                  'reason': set([('list_head', 'prev')])},
13702                 {'call': 'remap_file_pages',
13703                  'reason': set([('list_head', 'prev')])},
13704                 {'call': 'dup3', 'reason': set([('list_head', 'prev')])},
13705                 {'call': 'readlinkat',
13706                  'reason': set([('list_head', 'prev'),
13707                                 ('timespec', 'tv_nsec'),
13708                                 ('timespec', 'tv_sec')])},
13709                 {'call': 'io_getevents',
13710                  'reason': set([('list_head', 'prev'),
13711                                 ('timespec', 'tv_nsec'),
13712                                 ('timespec', 'tv_sec')])},
13713                 {'call': 'getppid', 'reason': set([('list_head', 'prev')])},
13714                 {'call': 'fchown',
13715                  'reason': set([('list_head', 'prev'),
13716                                 ('timespec', 'tv_nsec'),
13717                                 ('timespec', 'tv_sec')])},
13718                 {'call': 'mq_timedreceive',
13719                  'reason': set([('list_head', 'prev'),
13720                                 ('timespec', 'tv_nsec'),
13721                                 ('timespec', 'tv_sec')])},
```

```
13722                 {'call': 'capget', 'reason': set([('list_head', 'prev')])},
13723                 {'call': 'utime',
13724                  'reason': set([('timespec', 'tv_nsec'),
13725                                 ('timespec', 'tv_sec')])},
13726                 {'call': 'sched_setaffinity',
13727                  'reason': set([('list_head', 'prev')])},
13728                 {'call': 'ustat', 'reason': set([('list_head', 'prev')])},
13729                 {'call': 'bpf', 'reason': set([('list_head', 'prev')])},
13730                 {'call': 'unshare', 'reason': set([('list_head', 'prev')])},
13731                 {'call': 'signal', 'reason': set([('list_head', 'prev')])},
13732                 {'call': 'setreuid', 'reason': set([('list_head', 'prev')])},
13733                 {'call': 'umount', 'reason': set([('list_head', 'prev')])},
13734                 {'call': 'settimeofday',
13735                  'reason': set([('timespec', 'tv_nsec'),
13736                                 ('timespec', 'tv_sec')])},
13737                 {'call': 'timer_create',
13738                  'reason': set([('list_head', 'prev')])},
13739                 {'call': 'mkdirat', 'reason': set([('list_head', 'prev')])},
13740                 {'call': 'sched_rr_get_interval',
13741                  'reason': set([('list_head', 'prev'),
13742                                 ('timespec', 'tv_nsec'),
13743                                 ('timespec', 'tv_sec')])},
13744                 {'call': 'epoll_create1',
13745                  'reason': set([('list_head', 'prev')])},
13746                 {'call': 'timerfd_gettime',
13747                  'reason': set([('list_head', 'prev'),
13748                                 ('timespec', 'tv_nsec'),
13749                                 ('timespec', 'tv_sec')])},
13750                 {'call': 'tee', 'reason': set([('list_head', 'prev')])},
13751                 {'call': 'semctl',
13752                  'reason': set([('kern_ipc_perm', 'deleted'),
13753                                 ('list_head', 'prev'),
13754                                 ('sem_array', 'complex_count'),
13755                                 ('sem_array', 'sem_nsems'),
13756                                 ('sem_array', 'use_global_lock'),
13757                                 ('sem_undo', 'semid')])},
13758                 {'call': 'sync_file_range',
13759                  'reason': set([('list_head', 'prev')])},
13760                 {'call': 'epoll_ctl', 'reason': set([('list_head', 'prev')])},
13761                 {'call': 'flock', 'reason': set([('list_head', 'prev')])},
13762                 {'call': 'openat', 'reason': set([('list_head', 'prev')])},
13763                 {'call': 'lookup_dcookie',
13764                  'reason': set([('list_head', 'prev')])},
13765                 {'call': 'pselect6',
13766                  'reason': set([('timespec', 'tv_nsec'),
13767                                 ('timespec', 'tv_sec')])},
13768                 {'call': 'uselib',
13769                  'reason': set([('list_head', 'prev'),
13770                                 ('timespec', 'tv_nsec'),
13771                                 ('timespec', 'tv_sec')])},
13772                 {'call': 'renameat2', 'reason': set([('list_head', 'prev')])},
13773                 {'call': 'rt_sigprocmask',
13774                  'reason': set([('list_head', 'prev')])},
13775                 {'call': 'accept4', 'reason': set([('list_head', 'prev')])},
13776                 {'call': 'msgctl',
13777                  'reason': set([('kern_ipc_perm', 'deleted'),
13778                                 ('list_head', 'prev')])},
13779                 {'call': 'reboot', 'reason': set([('list_head', 'prev')])},
13780                 {'call': 'setsid', 'reason': set([('list_head', 'prev')])},
13781                 {'call': 'set_trip_temp',
13782                  'reason': set([('list_head', 'prev')])},
13783                 {'call': 'sigaltstack',
13784                  'reason': set([('list_head', 'prev')])},
13785                 {'call': 'sched_setattr',
13786                  'reason': set([('list_head', 'prev')])},
13787                 {'call': 'inotify_rm_watch',
```



```
13788                  'reason': set([('list_head', 'prev')])},
13789                 {'call': 'socketpair',
13790                  'reason': set([('list_head', 'prev')])},
13791                 {'call': 'migrate_pages',
13792                  'reason': set([('list_head', 'prev')])},
13793                 {'call': 'getitimer', 'reason': set([('list_head', 'prev')])},
13794                 {'call': 'fchmodat',
13795                  'reason': set([('list_head', 'prev'),
13796                                 ('timespec', 'tv_nsec'),
13797                                 ('timespec', 'tv_sec')])},
13798                 {'call': 'setpgid', 'reason': set([('list_head', 'prev')])},
13799                 {'call': 'init_module',
13800                  'reason': set([('list_head', 'prev')])},
13801                 {'call': 'setresgid', 'reason': set([('list_head', 'prev')])},
13802                 {'call': 'getcwd', 'reason': set([('list_head', 'prev')])},
13803                 {'call': 'inotify_add_watch',
13804                  'reason': set([('list_head', 'prev'),
13805                                 ('timespec', 'tv_nsec'),
13806                                 ('timespec', 'tv_sec')])},
13807                 {'call': 'get_trip_temp',
13808                  'reason': set([('list_head', 'prev')])},
13809                 {'call': 'timer_settime',
13810                  'reason': set([('list_head', 'prev'),
13811                                 ('timespec', 'tv_nsec'),
13812                                 ('timespec', 'tv_sec')])},
13813                 {'call': 'setregid', 'reason': set([('list_head', 'prev')])},
13814                 {'call': 'splice', 'reason': set([('list_head', 'prev')])},
13815                 {'call': 'ftruncate',
13816                  'reason': set([('list_head', 'prev'),
13817                                 ('timespec', 'tv_nsec'),
13818                                 ('timespec', 'tv_sec')])},
13819                 {'call': 'timer_gettime',
13820                  'reason': set([('list_head', 'prev'),
13821                                 ('timespec', 'tv_nsec'),
13822                                 ('timespec', 'tv_sec')])},
13823                 {'call': 'getsid', 'reason': set([('list_head', 'prev')])},
13824                 {'call': 'shmat',
13825                  'reason': set([('kern_ipc_perm', 'deleted'),
13826                                 ('list_head', 'prev')])},
13827                 {'call': 'mknodat', 'reason': set([('list_head', 'prev')])},
13828                 {'call': 'socket', 'reason': set([('list_head', 'prev')])},
13829                 {'call': 'symlinkat', 'reason': set([('list_head', 'prev')])},
13830                 {'call': 'pipe2', 'reason': set([('list_head', 'prev')])},
13831                 {'call': 'ioctl',
13832                  'reason': set([('list_head', 'prev'),
13833                                 ('timespec', 'tv_nsec'),
13834                                 ('timespec', 'tv_sec')])},
13835                 {'call': 'prlimit64', 'reason': set([('list_head', 'prev')])},
13836                 {'call': 'perf_event_open',
13837                  'reason': set([('list_head', 'prev')])},
13838                 {'call': 'linkat',
13839                  'reason': set([('list_head', 'prev'),
13840                                 ('timespec', 'tv_nsec'),
13841                                 ('timespec', 'tv_sec')])},
13842                 {'call': 'stime',
13843                  'reason': set([('timespec', 'tv_nsec'),
13844                                 ('timespec', 'tv_sec')])},
13845                 {'call': 'getgroups16',
13846                  'reason': set([('list_head', 'prev')])},
13847                 {'call': 'shmdt', 'reason': set([('list_head', 'prev')])},
13848                 {'call': 'quotactl', 'reason': set([('list_head', 'prev')])},
13849                 {'call': 'rt_sigaction',
13850                  'reason': set([('list_head', 'prev')])},
13851                 {'call': 'futimesat',
13852                  'reason': set([('timespec', 'tv_nsec'),
13853                                 ('timespec', 'tv_sec')])},
```

```
13854                 {'call': 'request_key',
13855                  'reason': set([('list_head', 'prev')])},
13856                 {'call': 'getpgid', 'reason': set([('list_head', 'prev')])},
13857                 {'call': 'brk', 'reason': set([('list_head', 'prev')])},
13858                 {'call': 'acct', 'reason': set([('list_head', 'prev')])},
13859                 {'call': 'poll',
13860                  'reason': set([('timespec', 'tv_nsec'),
13861                                 ('timespec', 'tv_sec')])},
13862                 {'call': 'open', 'reason': set([('list_head', 'prev')])},
13863                 {'call': 'select',
13864                  'reason': set([('timespec', 'tv_nsec'),
13865                                 ('timespec', 'tv_sec')])},
13866                 {'call': 'unlink',
13867                  'reason': set([('list_head', 'prev'),
13868                                 ('timespec', 'tv_nsec'),
13869                                 ('timespec', 'tv_sec')])},
13870                 {'call': 'exit_group',
13871                  'reason': set([('list_head', 'prev')])},
13872                 {'call': 'getpriority',
13873                  'reason': set([('list_head', 'prev')])},
13874                 {'call': 'sigaction', 'reason': set([('list_head', 'prev')])},
13875                 {'call': 'nanosleep',
13876                  'reason': set([('timespec', 'tv_nsec'),
13877                                 ('timespec', 'tv_sec')])},
13878                 {'call': 'mq_getsetattr',
13879                  'reason': set([('list_head', 'prev'),
13880                                 ('timespec', 'tv_nsec'),
13881                                 ('timespec', 'tv_sec')])},
13882                 {'call': 'faccessat',
13883                  'reason': set([('list_head', 'prev'),
13884                                 ('timespec', 'tv_nsec'),
13885                                 ('timespec', 'tv_sec')])},
13886                 {'call': 'rmdir', 'reason': set([('list_head', 'prev')])},
13887                 {'call': 'dup', 'reason': set([('list_head', 'prev')])},
13888                 {'call': 'setgroups16',
13889                  'reason': set([('list_head', 'prev')])},
13890                 {'call': 'setns', 'reason': set([('list_head', 'prev')])},
13891                 {'call': 'fork', 'reason': set([('list_head', 'prev')])},
13892                 {'call': 'get_mempolicy',
13893                  'reason': set([('list_head', 'prev')])},
13894                 {'call': 'io_submit', 'reason': set([('list_head', 'prev')])},
13895                 {'call': 'get_robust_list',
13896                  'reason': set([('list_head', 'prev')])},
13897                 {'call': 'mq_timedsend',
13898                  'reason': set([('list_head', 'prev'),
13899                                 ('timespec', 'tv_nsec'),
13900                                 ('timespec', 'tv_sec')])},
13901                 {'call': 'sched_yield',
13902                  'reason': set([('list_head', 'prev')])},
13903                 {'call': 'sched_getscheduler',
13904                  'reason': set([('list_head', 'prev')])},
13905                 {'call': 'ptrace', 'reason': set([('list_head', 'prev')])},
13906                 {'call': 'shmctl',
13907                  'reason': set([('kern_ipc_perm', 'deleted'),
13908                                 ('list_head', 'prev')])},
13909                 {'call': 'munlockall',
13910                  'reason': set([('list_head', 'prev')])},
13911                 {'call': 'swapon',
13912                  'reason': set([('list_head', 'prev'),
13913                                 ('timespec', 'tv_nsec'),
13914                                 ('timespec', 'tv_sec')])},
13915                 {'call': 'pkey_mprotect',
13916                  'reason': set([('list_head', 'prev')])},
13917                 {'call': 'madvise', 'reason': set([('list_head', 'prev')])},
13918                 {'call': 'epoll_wait',
13919                  'reason': set([('list_head', 'prev'),
```



```
13920                                 ('timespec', 'tv_nsec'),
13921                                 ('timespec', 'tv_sec')])},
13922                 {'call': 'sched_getattr',
13923                  'reason': set([('list_head', 'prev')])},
13924                 {'call': 'fchownat',
13925                  'reason': set([('list_head', 'prev'),
13926                                 ('timespec', 'tv_nsec'),
13927                                 ('timespec', 'tv_sec')])},
13928                 {'call': 'getrusage', 'reason': set([('list_head', 'prev')])},
13929                 {'call': 'fstat',
13930                  'reason': set([('timespec', 'tv_nsec'),
13931                                 ('timespec', 'tv_sec')])},
13932                 {'call': 'timerfd_settime',
13933                  'reason': set([('list_head', 'prev'),
13934                                 ('timespec', 'tv_nsec'),
13935                                 ('timespec', 'tv_sec')])},
13936                 {'call': 'sched_setscheduler',
13937                  'reason': set([('list_head', 'prev')])},
13938                 {'call': 'setresuid', 'reason': set([('list_head', 'prev')])},
13939                 {'call': 'setitimer', 'reason': set([('list_head', 'prev')])},
13940                 {'call': 'ioprio_get',
13941                  'reason': set([('list_head', 'prev')])},
13942                 {'call': 'vfork', 'reason': set([('list_head', 'prev')])},
13943                 {'call': 'setuid', 'reason': set([('list_head', 'prev')])},
13944                 {'call': 'io_setup', 'reason': set([('list_head', 'prev')])},
13945                 {'call': 'mprotect', 'reason': set([('list_head', 'prev')])},
13946                 {'call': 'mmap_pgoff',
13947                  'reason': set([('list_head', 'prev')])},
13948                 {'call': 'mremap', 'reason': set([('list_head', 'prev')])},
13949                 {'call': 'io_destroy',
13950                  'reason': set([('list_head', 'prev')])},
13951                 {'call': 'mbind', 'reason': set([('list_head', 'prev')])},
13952                 {'call': 'prctl', 'reason': set([('list_head', 'prev')])},
13953                 {'call': 'move_pages',
13954                  'reason': set([('list_head', 'prev')])},
13955                 {'call': 'timerfd_create',
13956                  'reason': set([('list_head', 'prev')])},
13957                 {'call': 'modify_ldt',
13958                  'reason': set([('list_head', 'prev')])},
13959                 {'call': 'getgroups', 'reason': set([('list_head', 'prev')])},
13960                 {'call': 'dup2', 'reason': set([('list_head', 'prev')])},
13961                 {'call': 'get_curr_temp',
13962                  'reason': set([('list_head', 'prev')])},
13963                 {'call': 'msgsnd',
13964                  'reason': set([('kern_ipc_perm', 'deleted'),
13965                                 ('list_head', 'prev')])},
13966                 {'call': 'munlock', 'reason': set([('list_head', 'prev')])},
13967                 {'call': 'setpriority',
13968                  'reason': set([('list_head', 'prev')])},
13969                 {'call': 'inotify_init1',
13970                  'reason': set([('list_head', 'prev')])},
13971                 {'call': 'mincore', 'reason': set([('list_head', 'prev')])},
13972                 {'call': 'mq_notify',
13973                  'reason': set([('list_head', 'prev'),
13974                                 ('timespec', 'tv_nsec'),
13975                                 ('timespec', 'tv_sec')])},
13976                 {'call': 'sendfile',
13977                  'reason': set([('list_head', 'prev'),
13978                                 ('timespec', 'tv_nsec'),
13979                                 ('timespec', 'tv_sec')])},
13980                 {'call': 'timer_getoverrun',
13981                  'reason': set([('list_head', 'prev')])},
13982                 {'call': 'newfstat',
13983                  'reason': set([('timespec', 'tv_nsec'),
13984                                 ('timespec', 'tv_sec')])},
13985                 {'call': 'kexec_load',
```

```
                 'reason': set([('list_head', 'prev')])},
                {'call': 'clone', 'reason': set([('list_head', 'prev')])},
                {'call': 'mq_open', 'reason': set([('list_head', 'prev')])},
                {'call': 'setgroups', 'reason': set([('list_head', 'prev')])},
                {'call': 'clock_nanosleep',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'unlinkat',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'sched_getparam',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'io_cancel', 'reason': set([('list_head', 'prev')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'futex',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'recvmmsg',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'finit_module',
                 'reason': set([('list_head', 'prev')])},
                {'call': 'sendfile64',
                 'reason': set([('list_head', 'prev'),
                                ('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'mlockall', 'reason': set([('list_head', 'prev')])},
                {'call': 'ppoll',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])}],
 'sendfile': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
              {'call': 'remap_file_pages',
               'reason': set([('file', 'f_mode')])},
              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
              {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
              {'call': 'flock', 'reason': set([('file', 'f_mode')])},
              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
              {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
              {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
              {'call': 'socket', 'reason': set([('file', 'f_mode')])},
              {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
              {'call': 'perf_event_open',
               'reason': set([('file', 'f_mode')])},
              {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
              {'call': 'acct', 'reason': set([('file', 'f_mode')])},
              {'call': 'open', 'reason': set([('file', 'f_mode')])},
              {'call': 'dup', 'reason': set([('file', 'f_mode')])},
              {'call': 'setns', 'reason': set([('file', 'f_mode')])},
              {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
              {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
              {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
              {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
              {'call': 'open_by_handle_at',
               'reason': set([('file', 'f_mode')])}],
 'sendfile64': [{'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
                {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
                {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
                {'call': 'remap_file_pages',
                 'reason': set([('file', 'f_mode')])},
```



```
                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
                {'call': 'epoll_create1',
                 'reason': set([('file', 'f_mode')])},
                {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
                {'call': 'flock', 'reason': set([('file', 'f_mode')])},
                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
                {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
                {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
                {'call': 'perf_event_open',
                 'reason': set([('file', 'f_mode')])},
                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
                {'call': 'open', 'reason': set([('file', 'f_mode')])},
                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_mode')])}],
 'sendmmsg': [{'call': 'recvfrom', 'reason': set([('socket', 'file')])},
              {'call': 'sendto', 'reason': set([('socket', 'file')])},
              {'call': 'connect', 'reason': set([('socket', 'file')])},
              {'call': 'getsockname', 'reason': set([('socket', 'file')])},
              {'call': 'accept4', 'reason': set([('socket', 'file')])},
              {'call': 'getpeername', 'reason': set([('socket', 'file')])},
              {'call': 'setsockopt', 'reason': set([('socket', 'file')])},
              {'call': 'sendmsg', 'reason': set([('socket', 'file')])},
              {'call': 'shutdown', 'reason': set([('socket', 'file')])},
              {'call': 'getsockopt', 'reason': set([('socket', 'file')])},
              {'call': 'listen', 'reason': set([('socket', 'file')])},
              {'call': 'recvmsg', 'reason': set([('socket', 'file')])},
              {'call': 'bind', 'reason': set([('socket', 'file')])},
              {'call': 'recvmmsg', 'reason': set([('socket', 'file')])}],
 'sendmsg': [{'call': 'recvfrom', 'reason': set([('socket', 'file')])},
             {'call': 'sendto', 'reason': set([('socket', 'file')])},
             {'call': 'connect', 'reason': set([('socket', 'file')])},
             {'call': 'getsockname', 'reason': set([('socket', 'file')])},
             {'call': 'accept4', 'reason': set([('socket', 'file')])},
             {'call': 'getpeername', 'reason': set([('socket', 'file')])},
             {'call': 'setsockopt', 'reason': set([('socket', 'file')])},
             {'call': 'shutdown', 'reason': set([('socket', 'file')])},
             {'call': 'getsockopt', 'reason': set([('socket', 'file')])},
             {'call': 'listen', 'reason': set([('socket', 'file')])},
             {'call': 'recvmsg', 'reason': set([('socket', 'file')])},
             {'call': 'sendmmsg', 'reason': set([('socket', 'file')])},
             {'call': 'bind', 'reason': set([('socket', 'file')])},
             {'call': 'recvmmsg', 'reason': set([('socket', 'file')])}],
 'sendto': [{'call': 'eventfd2', 'reason': set([('file', 'f_flags')])},
            {'call': 'swapoff', 'reason': set([('file', 'f_flags')])},
            {'call': 'memfd_create', 'reason': set([('file', 'f_flags')])},
            {'call': 'remap_file_pages',
             'reason': set([('file', 'f_flags')])},
            {'call': 'dup3', 'reason': set([('file', 'f_flags')])},
            {'call': 'epoll_create1', 'reason': set([('file', 'f_flags')])},
            {'call': 'epoll_ctl', 'reason': set([('file', 'f_flags')])},
            {'call': 'flock', 'reason': set([('file', 'f_flags')])},
            {'call': 'openat', 'reason': set([('file', 'f_flags')])},
            {'call': 'uselib', 'reason': set([('file', 'f_flags')])},
            {'call': 'accept4', 'reason': set([('file', 'f_flags')])},
            {'call': 'socketpair', 'reason': set([('file', 'f_flags')])},
```

```
14118             {'call': 'shmat', 'reason': set([('file', 'f_flags')])},
14119             {'call': 'socket', 'reason': set([('file', 'f_flags')])},
14120             {'call': 'pipe2', 'reason': set([('file', 'f_flags')])},
14121             {'call': 'perf_event_open', 'reason': set([('file', 'f_flags')])},
14122             {'call': 'shmdt', 'reason': set([('file', 'f_flags')])},
14123             {'call': 'acct', 'reason': set([('file', 'f_flags')])},
14124             {'call': 'open', 'reason': set([('file', 'f_flags')])},
14125             {'call': 'mq_getsetattr', 'reason': set([('file', 'f_flags')])},
14126             {'call': 'dup', 'reason': set([('file', 'f_flags')])},
14127             {'call': 'setns', 'reason': set([('file', 'f_flags')])},
14128             {'call': 'shmctl', 'reason': set([('file', 'f_flags')])},
14129             {'call': 'swapon', 'reason': set([('file', 'f_flags')])},
14130             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_flags')])},
14131             {'call': 'mq_open', 'reason': set([('file', 'f_flags')])},
14132             {'call': 'open_by_handle_at',
14133              'reason': set([('file', 'f_flags')])}],
14134  'set_mempolicy': [{'call': 'get_mempolicy',
14135                     'reason': set([('mempolicy', 'mode')])},
14136                    {'call': 'mbind', 'reason': set([('mempolicy', 'mode')])}],
14137  'set_thread_area': [{'call': 'keyctl',
14138                       'reason': set([('thread_struct', 'fsindex'),
14139                                      ('thread_struct', 'gsindex')])},
14140                      {'call': 'rt_sigtimedwait',
14141                       'reason': set([('thread_struct', 'fsindex'),
14142                                      ('thread_struct', 'gsindex')])},
14143                      {'call': 'msgrcv',
14144                       'reason': set([('thread_struct', 'fsindex'),
14145                                      ('thread_struct', 'gsindex')])},
14146                      {'call': 'kill',
14147                       'reason': set([('thread_struct', 'fsindex'),
14148                                      ('thread_struct', 'gsindex')])},
14149                      {'call': 'sched_getaffinity',
14150                       'reason': set([('thread_struct', 'fsindex'),
14151                                      ('thread_struct', 'gsindex')])},
14152                      {'call': 'arch_prctl',
14153                       'reason': set([('thread_struct', 'fsindex'),
14154                                      ('thread_struct', 'gsindex')])},
14155                      {'call': 'sched_setparam',
14156                       'reason': set([('thread_struct', 'fsindex'),
14157                                      ('thread_struct', 'gsindex')])},
14158                      {'call': 'ioprio_set',
14159                       'reason': set([('thread_struct', 'fsindex'),
14160                                      ('thread_struct', 'gsindex')])},
14161                      {'call': 'getppid',
14162                       'reason': set([('thread_struct', 'fsindex'),
14163                                      ('thread_struct', 'gsindex')])},
14164                      {'call': 'ioperm',
14165                       'reason': set([('thread_struct', 'fsindex'),
14166                                      ('thread_struct', 'gsindex')])},
14167                      {'call': 'mq_timedreceive',
14168                       'reason': set([('thread_struct', 'fsindex'),
14169                                      ('thread_struct', 'gsindex')])},
14170                      {'call': 'capget',
14171                       'reason': set([('thread_struct', 'fsindex'),
14172                                      ('thread_struct', 'gsindex')])},
14173                      {'call': 'sched_setaffinity',
14174                       'reason': set([('thread_struct', 'fsindex'),
14175                                      ('thread_struct', 'gsindex')])},
14176                      {'call': 'signal',
14177                       'reason': set([('thread_struct', 'fsindex'),
14178                                      ('thread_struct', 'gsindex')])},
14179                      {'call': 'semtimedop',
14180                       'reason': set([('thread_struct', 'fsindex'),
14181                                      ('thread_struct', 'gsindex')])},
14182                      {'call': 'umount',
14183                       'reason': set([('thread_struct', 'fsindex'),
```



```
14184                                      ('thread_struct', 'gsindex')])},
14185                      {'call': 'sched_rr_get_interval',
14186                       'reason': set([('thread_struct', 'fsindex'),
14187                                      ('thread_struct', 'gsindex')])},
14188                      {'call': 'rt_sigprocmask',
14189                       'reason': set([('thread_struct', 'fsindex'),
14190                                      ('thread_struct', 'gsindex')])},
14191                      {'call': 'setsid',
14192                       'reason': set([('thread_struct', 'fsindex'),
14193                                      ('thread_struct', 'gsindex')])},
14194                      {'call': 'sigaltstack',
14195                       'reason': set([('thread_struct', 'fsindex'),
14196                                      ('thread_struct', 'gsindex')])},
14197                      {'call': 'sched_setattr',
14198                       'reason': set([('thread_struct', 'fsindex'),
14199                                      ('thread_struct', 'gsindex')])},
14200                      {'call': 'migrate_pages',
14201                       'reason': set([('thread_struct', 'fsindex'),
14202                                      ('thread_struct', 'gsindex')])},
14203                      {'call': 'getitimer',
14204                       'reason': set([('thread_struct', 'fsindex'),
14205                                      ('thread_struct', 'gsindex')])},
14206                      {'call': 'setpgid',
14207                       'reason': set([('thread_struct', 'fsindex'),
14208                                      ('thread_struct', 'gsindex')])},
14209                      {'call': 'getsid',
14210                       'reason': set([('thread_struct', 'fsindex'),
14211                                      ('thread_struct', 'gsindex')])},
14212                      {'call': 'prlimit64',
14213                       'reason': set([('thread_struct', 'fsindex'),
14214                                      ('thread_struct', 'gsindex')])},
14215                      {'call': 'perf_event_open',
14216                       'reason': set([('thread_struct', 'fsindex'),
14217                                      ('thread_struct', 'gsindex')])},
14218                      {'call': 'rt_sigaction',
14219                       'reason': set([('thread_struct', 'fsindex'),
14220                                      ('thread_struct', 'gsindex')])},
14221                      {'call': 'getpgid',
14222                       'reason': set([('thread_struct', 'fsindex'),
14223                                      ('thread_struct', 'gsindex')])},
14224                      {'call': 'getpriority',
14225                       'reason': set([('thread_struct', 'fsindex'),
14226                                      ('thread_struct', 'gsindex')])},
14227                      {'call': 'sigaction',
14228                       'reason': set([('thread_struct', 'fsindex'),
14229                                      ('thread_struct', 'gsindex')])},
14230                      {'call': 'setns',
14231                       'reason': set([('thread_struct', 'fsindex'),
14232                                      ('thread_struct', 'gsindex')])},
14233                      {'call': 'fork',
14234                       'reason': set([('thread_struct', 'fsindex'),
14235                                      ('thread_struct', 'gsindex')])},
14236                      {'call': 'get_robust_list',
14237                       'reason': set([('thread_struct', 'fsindex'),
14238                                      ('thread_struct', 'gsindex')])},
14239                      {'call': 'mq_timedsend',
14240                       'reason': set([('thread_struct', 'fsindex'),
14241                                      ('thread_struct', 'gsindex')])},
14242                      {'call': 'sched_getscheduler',
14243                       'reason': set([('thread_struct', 'fsindex'),
14244                                      ('thread_struct', 'gsindex')])},
14245                      {'call': 'ptrace',
14246                       'reason': set([('thread_struct', 'fsindex'),
14247                                      ('thread_struct', 'gsindex')])},
14248                      {'call': 'sched_getattr',
14249                       'reason': set([('thread_struct', 'fsindex'),
```

```
                                    ('thread_struct', 'gsindex')])},
                     {'call': 'getrusage',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'sched_setscheduler',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'setitimer',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'ioprio_get',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'vfork',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'prctl',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'move_pages',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'setpriority',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'clone',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])},
                     {'call': 'sched_getparam',
                      'reason': set([('thread_struct', 'fsindex'),
                                     ('thread_struct', 'gsindex')])}],
 'set_trip_temp': [{'call': 'get_trip_temp',
                    'reason': set([('pkg_device', 'cpu'),
                                   ('pkg_device', 'tj_max')])},
                   {'call': 'get_curr_temp',
                    'reason': set([('pkg_device', 'cpu'),
                                   ('pkg_device', 'tj_max')])}],
 'setdomainname': [{'call': 'setns',
                    'reason': set([('uts_namespace', 'user_ns')])}],
 'setfsgid': [{'call': 'keyctl',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'sched_getaffinity',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setparam',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setgid',
               'reason': set([('cred', 'egid'),
                              ('cred', 'fsgid'),
                              ('cred', 'gid'),
                              ('cred', 'sgid'),
                              ('cred', 'user_ns')])},
              {'call': 'ioprio_set',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'mq_timedreceive',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
```



```
              {'call': 'sched_setaffinity',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'semtimedop',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sigaltstack',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setattr',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'migrate_pages',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setresgid',
               'reason': set([('cred', 'egid'),
                              ('cred', 'fsgid'),
                              ('cred', 'gid'),
                              ('cred', 'sgid'),
                              ('cred', 'user_ns')])},
              {'call': 'setregid',
               'reason': set([('cred', 'egid'),
                              ('cred', 'fsgid'),
                              ('cred', 'gid'),
                              ('cred', 'sgid'),
                              ('cred', 'user_ns')])},
              {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'prlimit64',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'perf_event_open',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
              {'call': 'rt_sigaction',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getpriority',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
              {'call': 'faccessat',
               'reason': set([('cred', 'fsgid'), ('cred', 'user_ns')])},
              {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
              {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
              {'call': 'get_robust_list',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'mq_timedsend',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_getscheduler',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_getattr',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setscheduler',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
              {'call': 'ioprio_get',
```

```
14382                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14383               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
14384               {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
14385               {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
14386               {'call': 'move_pages',
14387                'reason': set([('task_struct', 'cred')])},
14388               {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
14389               {'call': 'setpriority',
14390                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14391               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
14392               {'call': 'sched_getparam',
14393                'reason': set([('task_struct', 'cred')])}],
14394  'setfsuid': [{'call': 'keyctl',
14395                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14396               {'call': 'rt_sigtimedwait',
14397                'reason': set([('task_struct', 'cred')])},
14398               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
14399               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
14400               {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
14401               {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
14402               {'call': 'sched_getaffinity',
14403                'reason': set([('task_struct', 'cred')])},
14404               {'call': 'sched_setparam',
14405                'reason': set([('task_struct', 'cred')])},
14406               {'call': 'setgid', 'reason': set([('cred', 'user_ns')])},
14407               {'call': 'ioprio_set',
14408                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14409               {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
14410               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
14411               {'call': 'mq_timedreceive',
14412                'reason': set([('task_struct', 'cred')])},
14413               {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
14414               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
14415               {'call': 'sched_setaffinity',
14416                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14417               {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
14418               {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
14419               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
14420               {'call': 'setreuid',
14421                'reason': set([('cred', 'euid'),
14422                               ('cred', 'fsuid'),
14423                               ('cred', 'suid'),
14424                               ('cred', 'uid'),
14425                               ('cred', 'user_ns')])},
14426               {'call': 'semtimedop',
14427                'reason': set([('task_struct', 'cred')])},
14428               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
14429               {'call': 'sched_rr_get_interval',
14430                'reason': set([('task_struct', 'cred')])},
14431               {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
14432               {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
14433               {'call': 'rt_sigprocmask',
14434                'reason': set([('task_struct', 'cred')])},
14435               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
14436               {'call': 'sigaltstack',
14437                'reason': set([('task_struct', 'cred')])},
14438               {'call': 'sched_setattr',
14439                'reason': set([('task_struct', 'cred')])},
14440               {'call': 'migrate_pages',
14441                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14442               {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
14443               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
14444               {'call': 'setresgid', 'reason': set([('cred', 'user_ns')])},
14445               {'call': 'setregid', 'reason': set([('cred', 'user_ns')])},
14446               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
14447               {'call': 'prlimit64',
```



```
14448                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14449               {'call': 'perf_event_open',
14450                'reason': set([('task_struct', 'cred')])},
14451               {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
14452               {'call': 'rt_sigaction',
14453                'reason': set([('task_struct', 'cred')])},
14454               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
14455               {'call': 'getpriority',
14456                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14457               {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
14458               {'call': 'faccessat',
14459                'reason': set([('cred', 'fsuid'), ('cred', 'user_ns')])},
14460               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
14461               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
14462               {'call': 'get_robust_list',
14463                'reason': set([('task_struct', 'cred')])},
14464               {'call': 'mq_timedsend',
14465                'reason': set([('task_struct', 'cred')])},
14466               {'call': 'sched_getscheduler',
14467                'reason': set([('task_struct', 'cred')])},
14468               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
14469               {'call': 'sched_getattr',
14470                'reason': set([('task_struct', 'cred')])},
14471               {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
14472               {'call': 'sched_setscheduler',
14473                'reason': set([('task_struct', 'cred')])},
14474               {'call': 'setresuid',
14475                'reason': set([('cred', 'euid'),
14476                               ('cred', 'fsuid'),
14477                               ('cred', 'suid'),
14478                               ('cred', 'uid'),
14479                               ('cred', 'user_ns')])},
14480               {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
14481               {'call': 'ioprio_get',
14482                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14483               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
14484               {'call': 'setuid',
14485                'reason': set([('cred', 'euid'),
14486                               ('cred', 'fsuid'),
14487                               ('cred', 'suid'),
14488                               ('cred', 'uid'),
14489                               ('cred', 'user_ns')])},
14490               {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
14491               {'call': 'move_pages',
14492                'reason': set([('task_struct', 'cred')])},
14493               {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
14494               {'call': 'setpriority',
14495                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14496               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
14497               {'call': 'sched_getparam',
14498                'reason': set([('task_struct', 'cred')])}],
14499  'setgid': [{'call': 'keyctl',
14500              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14501             {'call': 'rt_sigtimedwait',
14502              'reason': set([('task_struct', 'cred')])},
14503             {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
14504             {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
14505             {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
14506             {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
14507             {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
14508             {'call': 'sched_getaffinity',
14509              'reason': set([('task_struct', 'cred')])},
14510             {'call': 'sched_setparam',
14511              'reason': set([('task_struct', 'cred')])},
14512             {'call': 'ioprio_set',
14513              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
```

```
14514             {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
14515             {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
14516             {'call': 'mq_timedreceive',
14517              'reason': set([('task_struct', 'cred')])},
14518             {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
14519             {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
14520             {'call': 'sched_setaffinity',
14521              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14522             {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
14523             {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
14524             {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
14525             {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
14526             {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
14527             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
14528             {'call': 'sched_rr_get_interval',
14529              'reason': set([('task_struct', 'cred')])},
14530             {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
14531             {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
14532             {'call': 'rt_sigprocmask',
14533              'reason': set([('task_struct', 'cred')])},
14534             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
14535             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
14536             {'call': 'sched_setattr',
14537              'reason': set([('task_struct', 'cred')])},
14538             {'call': 'migrate_pages',
14539              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14540             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
14541             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
14542             {'call': 'setresgid',
14543              'reason': set([('cred', 'gid'),
14544                             ('cred', 'sgid'),
14545                             ('cred', 'user_ns')])},
14546             {'call': 'setregid',
14547              'reason': set([('cred', 'gid'),
14548                             ('cred', 'sgid'),
14549                             ('cred', 'user_ns')])},
14550             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
14551             {'call': 'prlimit64',
14552              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14553             {'call': 'perf_event_open',
14554              'reason': set([('task_struct', 'cred')])},
14555             {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
14556             {'call': 'rt_sigaction',
14557              'reason': set([('task_struct', 'cred')])},
14558             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
14559             {'call': 'getpriority',
14560              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14561             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
14562             {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
14563             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
14564             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
14565             {'call': 'get_robust_list',
14566              'reason': set([('task_struct', 'cred')])},
14567             {'call': 'mq_timedsend',
14568              'reason': set([('task_struct', 'cred')])},
14569             {'call': 'sched_getscheduler',
14570              'reason': set([('task_struct', 'cred')])},
14571             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
14572             {'call': 'sched_getattr',
14573              'reason': set([('task_struct', 'cred')])},
14574             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
14575             {'call': 'sched_setscheduler',
14576              'reason': set([('task_struct', 'cred')])},
14577             {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
14578             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
14579             {'call': 'ioprio_get',
```



```
14580              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14581             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
14582             {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
14583             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
14584             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
14585             {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
14586             {'call': 'setpriority',
14587              'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
14588             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
14589             {'call': 'sched_getparam',
14590              'reason': set([('task_struct', 'cred')])}],
14591  'setgroups16': [{'call': 'setgroups',
14592                   'reason': set([('group_info', 'ngroups')])}],
14593  'sethostname': [{'call': 'setns',
14594                   'reason': set([('uts_namespace', 'user_ns')])}],
14595  'setitimer': [{'call': 'waitid',
14596                 'reason': set([('timeval', 'tv_sec'),
14597                                ('timeval', 'tv_usec')])},
14598                {'call': 'settimeofday',
14599                 'reason': set([('timeval', 'tv_sec'),
14600                                ('timeval', 'tv_usec')])},
14601                {'call': 'timer_create',
14602                 'reason': set([('signal_struct', 'it_real_incr')])},
14603                {'call': 'adjtimex',
14604                 'reason': set([('timeval', 'tv_sec'),
14605                                ('timeval', 'tv_usec')])},
14606                {'call': 'getitimer',
14607                 'reason': set([('itimerval', 'it_interval'),
14608                                ('itimerval', 'it_value'),
14609                                ('timeval', 'tv_sec'),
14610                                ('timeval', 'tv_usec')])},
14611                {'call': 'select',
14612                 'reason': set([('timeval', 'tv_sec'),
14613                                ('timeval', 'tv_usec')])},
14614                {'call': 'exit_group',
14615                 'reason': set([('signal_struct', 'it_real_incr')])},
14616                {'call': 'wait4',
14617                 'reason': set([('timeval', 'tv_sec'),
14618                                ('timeval', 'tv_usec')])},
14619                {'call': 'getrusage',
14620                 'reason': set([('timeval', 'tv_sec'),
14621                                ('timeval', 'tv_usec')])},
14622                {'call': 'clock_adjtime',
14623                 'reason': set([('timeval', 'tv_sec'),
14624                                ('timeval', 'tv_usec')])},
14625                {'call': 'alarm',
14626                 'reason': set([('timeval', 'tv_sec'),
14627                                ('timeval', 'tv_usec')])},
14628                {'call': 'ppoll',
14629                 'reason': set([('timeval', 'tv_sec'),
14630                                ('timeval', 'tv_usec')])}],
14631  'setpgid': [{'call': 'keyctl',
14632               'reason': set([('task_struct', 'exit_signal'),
14633                              ('task_struct', 'flags'),
14634                              ('task_struct', 'real_parent'),
14635                              ('task_struct', 'signal')])},
14636              {'call': 'rt_sigtimedwait',
14637               'reason': set([('task_struct', 'exit_signal'),
14638                              ('task_struct', 'flags'),
14639                              ('task_struct', 'real_parent'),
14640                              ('task_struct', 'signal')])},
14641              {'call': 'msgrcv',
14642               'reason': set([('task_struct', 'exit_signal'),
14643                              ('task_struct', 'flags'),
14644                              ('task_struct', 'real_parent'),
14645                              ('task_struct', 'signal')])},
```

```
14646             {'call': 'kill',
14647              'reason': set([('task_struct', 'exit_signal'),
14648                             ('task_struct', 'flags'),
14649                             ('task_struct', 'real_parent'),
14650                             ('task_struct', 'signal')])},
14651             {'call': 'sched_getaffinity',
14652              'reason': set([('task_struct', 'exit_signal'),
14653                             ('task_struct', 'flags'),
14654                             ('task_struct', 'real_parent'),
14655                             ('task_struct', 'signal')])},
14656             {'call': 'sched_setparam',
14657              'reason': set([('task_struct', 'exit_signal'),
14658                             ('task_struct', 'flags'),
14659                             ('task_struct', 'real_parent'),
14660                             ('task_struct', 'signal')])},
14661             {'call': 'ioprio_set',
14662              'reason': set([('task_struct', 'exit_signal'),
14663                             ('task_struct', 'flags'),
14664                             ('task_struct', 'real_parent'),
14665                             ('task_struct', 'signal')])},
14666             {'call': 'getppid',
14667              'reason': set([('task_struct', 'exit_signal'),
14668                             ('task_struct', 'flags'),
14669                             ('task_struct', 'real_parent'),
14670                             ('task_struct', 'signal')])},
14671             {'call': 'mq_timedreceive',
14672              'reason': set([('task_struct', 'exit_signal'),
14673                             ('task_struct', 'flags'),
14674                             ('task_struct', 'real_parent'),
14675                             ('task_struct', 'signal')])},
14676             {'call': 'capget',
14677              'reason': set([('task_struct', 'exit_signal'),
14678                             ('task_struct', 'flags'),
14679                             ('task_struct', 'real_parent'),
14680                             ('task_struct', 'signal')])},
14681             {'call': 'sched_setaffinity',
14682              'reason': set([('task_struct', 'exit_signal'),
14683                             ('task_struct', 'flags'),
14684                             ('task_struct', 'real_parent'),
14685                             ('task_struct', 'signal')])},
14686             {'call': 'signal',
14687              'reason': set([('task_struct', 'exit_signal'),
14688                             ('task_struct', 'flags'),
14689                             ('task_struct', 'real_parent'),
14690                             ('task_struct', 'signal')])},
14691             {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
14692             {'call': 'semtimedop',
14693              'reason': set([('task_struct', 'exit_signal'),
14694                             ('task_struct', 'flags'),
14695                             ('task_struct', 'real_parent'),
14696                             ('task_struct', 'signal')])},
14697             {'call': 'umount',
14698              'reason': set([('task_struct', 'exit_signal'),
14699                             ('task_struct', 'flags'),
14700                             ('task_struct', 'real_parent'),
14701                             ('task_struct', 'signal')])},
14702             {'call': 'timer_create',
14703              'reason': set([('signal_struct', 'leader')])},
14704             {'call': 'sched_rr_get_interval',
14705              'reason': set([('task_struct', 'exit_signal'),
14706                             ('task_struct', 'flags'),
14707                             ('task_struct', 'real_parent'),
14708                             ('task_struct', 'signal')])},
14709             {'call': 'rt_sigprocmask',
14710              'reason': set([('task_struct', 'exit_signal'),
14711                             ('task_struct', 'flags'),
```



```
14712                             ('task_struct', 'real_parent'),
14713                             ('task_struct', 'signal')])},
14714             {'call': 'setsid',
14715              'reason': set([('signal_struct', 'leader'),
14716                             ('task_struct', 'exit_signal'),
14717                             ('task_struct', 'flags'),
14718                             ('task_struct', 'real_parent'),
14719                             ('task_struct', 'signal')])},
14720             {'call': 'sigaltstack',
14721              'reason': set([('task_struct', 'exit_signal'),
14722                             ('task_struct', 'flags'),
14723                             ('task_struct', 'real_parent'),
14724                             ('task_struct', 'signal')])},
14725             {'call': 'sched_setattr',
14726              'reason': set([('task_struct', 'exit_signal'),
14727                             ('task_struct', 'flags'),
14728                             ('task_struct', 'real_parent'),
14729                             ('task_struct', 'signal')])},
14730             {'call': 'migrate_pages',
14731              'reason': set([('task_struct', 'exit_signal'),
14732                             ('task_struct', 'flags'),
14733                             ('task_struct', 'real_parent'),
14734                             ('task_struct', 'signal')])},
14735             {'call': 'getitimer',
14736              'reason': set([('task_struct', 'exit_signal'),
14737                             ('task_struct', 'flags'),
14738                             ('task_struct', 'real_parent'),
14739                             ('task_struct', 'signal')])},
14740             {'call': 'getsid',
14741              'reason': set([('task_struct', 'exit_signal'),
14742                             ('task_struct', 'flags'),
14743                             ('task_struct', 'real_parent'),
14744                             ('task_struct', 'signal')])},
14745             {'call': 'prlimit64',
14746              'reason': set([('task_struct', 'exit_signal'),
14747                             ('task_struct', 'flags'),
14748                             ('task_struct', 'real_parent'),
14749                             ('task_struct', 'signal')])},
14750             {'call': 'perf_event_open',
14751              'reason': set([('task_struct', 'exit_signal'),
14752                             ('task_struct', 'flags'),
14753                             ('task_struct', 'real_parent'),
14754                             ('task_struct', 'signal')])},
14755             {'call': 'rt_sigaction',
14756              'reason': set([('task_struct', 'exit_signal'),
14757                             ('task_struct', 'flags'),
14758                             ('task_struct', 'real_parent'),
14759                             ('task_struct', 'signal')])},
14760             {'call': 'getpgid',
14761              'reason': set([('task_struct', 'exit_signal'),
14762                             ('task_struct', 'flags'),
14763                             ('task_struct', 'real_parent'),
14764                             ('task_struct', 'signal')])},
14765             {'call': 'exit_group',
14766              'reason': set([('signal_struct', 'leader')])},
14767             {'call': 'getpriority',
14768              'reason': set([('task_struct', 'exit_signal'),
14769                             ('task_struct', 'flags'),
14770                             ('task_struct', 'real_parent'),
14771                             ('task_struct', 'signal')])},
14772             {'call': 'sigaction',
14773              'reason': set([('task_struct', 'exit_signal'),
14774                             ('task_struct', 'flags'),
14775                             ('task_struct', 'real_parent'),
14776                             ('task_struct', 'signal')])},
14777             {'call': 'setns',
```

```
14778                'reason': set([('task_struct', 'exit_signal'),
14779                               ('task_struct', 'flags'),
14780                               ('task_struct', 'real_parent'),
14781                               ('task_struct', 'signal')])},
14782             {'call': 'fork',
14783              'reason': set([('task_struct', 'exit_signal'),
14784                             ('task_struct', 'flags'),
14785                             ('task_struct', 'real_parent'),
14786                             ('task_struct', 'signal')])},
14787             {'call': 'get_robust_list',
14788              'reason': set([('task_struct', 'exit_signal'),
14789                             ('task_struct', 'flags'),
14790                             ('task_struct', 'real_parent'),
14791                             ('task_struct', 'signal')])},
14792             {'call': 'mq_timedsend',
14793              'reason': set([('task_struct', 'exit_signal'),
14794                             ('task_struct', 'flags'),
14795                             ('task_struct', 'real_parent'),
14796                             ('task_struct', 'signal')])},
14797             {'call': 'sched_getscheduler',
14798              'reason': set([('task_struct', 'exit_signal'),
14799                             ('task_struct', 'flags'),
14800                             ('task_struct', 'real_parent'),
14801                             ('task_struct', 'signal')])},
14802             {'call': 'ptrace',
14803              'reason': set([('task_struct', 'exit_signal'),
14804                             ('task_struct', 'flags'),
14805                             ('task_struct', 'real_parent'),
14806                             ('task_struct', 'signal')])},
14807             {'call': 'sched_getattr',
14808              'reason': set([('task_struct', 'exit_signal'),
14809                             ('task_struct', 'flags'),
14810                             ('task_struct', 'real_parent'),
14811                             ('task_struct', 'signal')])},
14812             {'call': 'getrusage',
14813              'reason': set([('task_struct', 'exit_signal'),
14814                             ('task_struct', 'flags'),
14815                             ('task_struct', 'real_parent'),
14816                             ('task_struct', 'signal')])},
14817             {'call': 'sched_setscheduler',
14818              'reason': set([('task_struct', 'exit_signal'),
14819                             ('task_struct', 'flags'),
14820                             ('task_struct', 'real_parent'),
14821                             ('task_struct', 'signal')])},
14822             {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
14823             {'call': 'setitimer',
14824              'reason': set([('task_struct', 'exit_signal'),
14825                             ('task_struct', 'flags'),
14826                             ('task_struct', 'real_parent'),
14827                             ('task_struct', 'signal')])},
14828             {'call': 'ioprio_get',
14829              'reason': set([('task_struct', 'exit_signal'),
14830                             ('task_struct', 'flags'),
14831                             ('task_struct', 'real_parent'),
14832                             ('task_struct', 'signal')])},
14833             {'call': 'vfork',
14834              'reason': set([('task_struct', 'exit_signal'),
14835                             ('task_struct', 'flags'),
14836                             ('task_struct', 'real_parent'),
14837                             ('task_struct', 'signal')])},
14838             {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
14839             {'call': 'prctl',
14840              'reason': set([('task_struct', 'exit_signal'),
14841                             ('task_struct', 'flags'),
14842                             ('task_struct', 'real_parent'),
14843                             ('task_struct', 'signal')])},
```



```
14844             {'call': 'move_pages',
14845              'reason': set([('task_struct', 'exit_signal'),
14846                             ('task_struct', 'flags'),
14847                             ('task_struct', 'real_parent'),
14848                             ('task_struct', 'signal')])},
14849             {'call': 'setpriority',
14850              'reason': set([('task_struct', 'exit_signal'),
14851                             ('task_struct', 'flags'),
14852                             ('task_struct', 'real_parent'),
14853                             ('task_struct', 'signal')])},
14854             {'call': 'clone',
14855              'reason': set([('task_struct', 'exit_signal'),
14856                             ('task_struct', 'flags'),
14857                             ('task_struct', 'real_parent'),
14858                             ('task_struct', 'signal')])},
14859             {'call': 'sched_getparam',
14860              'reason': set([('task_struct', 'exit_signal'),
14861                             ('task_struct', 'flags'),
14862                             ('task_struct', 'real_parent'),
14863                             ('task_struct', 'signal')])}],
14864  'setpriority': [{'call': 'keyctl',
14865                   'reason': set([('task_struct', 'cred'),
14866                                  ('task_struct', 'real_cred')])},
14867                  {'call': 'rt_sigtimedwait',
14868                   'reason': set([('task_struct', 'cred'),
14869                                  ('task_struct', 'real_cred')])},
14870                  {'call': 'msgrcv',
14871                   'reason': set([('task_struct', 'cred'),
14872                                  ('task_struct', 'real_cred')])},
14873                  {'call': 'kill',
14874                   'reason': set([('task_struct', 'cred'),
14875                                  ('task_struct', 'real_cred')])},
14876                  {'call': 'sched_getaffinity',
14877                   'reason': set([('task_struct', 'cred'),
14878                                  ('task_struct', 'real_cred')])},
14879                  {'call': 'sched_setparam',
14880                   'reason': set([('task_struct', 'cred'),
14881                                  ('task_struct', 'real_cred')])},
14882                  {'call': 'ioprio_set',
14883                   'reason': set([('task_struct', 'cred'),
14884                                  ('task_struct', 'real_cred')])},
14885                  {'call': 'getppid',
14886                   'reason': set([('task_struct', 'cred'),
14887                                  ('task_struct', 'real_cred')])},
14888                  {'call': 'mq_timedreceive',
14889                   'reason': set([('task_struct', 'cred'),
14890                                  ('task_struct', 'real_cred')])},
14891                  {'call': 'capget',
14892                   'reason': set([('task_struct', 'cred'),
14893                                  ('task_struct', 'real_cred')])},
14894                  {'call': 'sched_setaffinity',
14895                   'reason': set([('task_struct', 'cred'),
14896                                  ('task_struct', 'real_cred')])},
14897                  {'call': 'signal',
14898                   'reason': set([('task_struct', 'cred'),
14899                                  ('task_struct', 'real_cred')])},
14900                  {'call': 'setreuid', 'reason': set([('cred', 'uid')])},
14901                  {'call': 'semtimedop',
14902                   'reason': set([('task_struct', 'cred'),
14903                                  ('task_struct', 'real_cred')])},
14904                  {'call': 'umount',
14905                   'reason': set([('task_struct', 'cred'),
14906                                  ('task_struct', 'real_cred')])},
14907                  {'call': 'sched_rr_get_interval',
14908                   'reason': set([('task_struct', 'cred'),
14909                                  ('task_struct', 'real_cred')])},
```

```
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
```



```
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setresuid', 'reason': set([('cred', 'uid')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'setuid', 'reason': set([('cred', 'uid')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'cred'),
                                ('task_struct', 'real_cred')])}],
 'setregid': [{'call': 'keyctl',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
              {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'sched_getaffinity',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setparam',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'setgid',
               'reason': set([('cred', 'egid'),
                              ('cred', 'gid'),
                              ('cred', 'sgid'),
                              ('cred', 'user_ns')])},
              {'call': 'ioprio_set',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
              {'call': 'mq_timedreceive',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_setaffinity',
               'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
              {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
              {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'semtimedop',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('task_struct', 'cred')])},
              {'call': 'epoll_create1', 'reason': set([('cred', 'user_ns')])},
              {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
              {'call': 'rt_sigprocmask',
```

```
15042                'reason': set([('task_struct', 'cred')])},
15043               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
15044               {'call': 'sigaltstack',
15045                'reason': set([('task_struct', 'cred')])},
15046               {'call': 'sched_setattr',
15047                'reason': set([('task_struct', 'cred')])},
15048               {'call': 'migrate_pages',
15049                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
15050               {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
15051               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
15052               {'call': 'setresgid',
15053                'reason': set([('cred', 'egid'),
15054                               ('cred', 'gid'),
15055                               ('cred', 'sgid'),
15056                               ('cred', 'user_ns')])},
15057               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
15058               {'call': 'prlimit64',
15059                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
15060               {'call': 'perf_event_open',
15061                'reason': set([('task_struct', 'cred')])},
15062               {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
15063               {'call': 'rt_sigaction',
15064                'reason': set([('task_struct', 'cred')])},
15065               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
15066               {'call': 'getpriority',
15067                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
15068               {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
15069               {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
15070               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
15071               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
15072               {'call': 'get_robust_list',
15073                'reason': set([('task_struct', 'cred')])},
15074               {'call': 'mq_timedsend',
15075                'reason': set([('task_struct', 'cred')])},
15076               {'call': 'sched_getscheduler',
15077                'reason': set([('task_struct', 'cred')])},
15078               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
15079               {'call': 'sched_getattr',
15080                'reason': set([('task_struct', 'cred')])},
15081               {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
15082               {'call': 'sched_setscheduler',
15083                'reason': set([('task_struct', 'cred')])},
15084               {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
15085               {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
15086               {'call': 'ioprio_get',
15087                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
15088               {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
15089               {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
15090               {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
15091               {'call': 'move_pages',
15092                'reason': set([('task_struct', 'cred')])},
15093               {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
15094               {'call': 'setpriority',
15095                'reason': set([('cred', 'user_ns'), ('task_struct', 'cred')])},
15096               {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
15097               {'call': 'sched_getparam',
15098                'reason': set([('task_struct', 'cred')])}],
15099  'setresgid': [{'call': 'keyctl',
15100                 'reason': set([('cred', 'user_ns'),
15101                                ('task_struct', 'cred')])},
15102                {'call': 'rt_sigtimedwait',
15103                 'reason': set([('task_struct', 'cred')])},
15104                {'call': 'setfsuid', 'reason': set([('cred', 'user_ns')])},
15105                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
15106                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
15107                {'call': 'getresuid16', 'reason': set([('cred', 'user_ns')])},
```



```
15108               {'call': 'getresgid', 'reason': set([('cred', 'user_ns')])},
15109               {'call': 'sched_getaffinity',
15110                'reason': set([('task_struct', 'cred')])},
15111               {'call': 'sched_setparam',
15112                'reason': set([('task_struct', 'cred')])},
15113               {'call': 'setgid',
15114                'reason': set([('cred', 'egid'),
15115                               ('cred', 'gid'),
15116                               ('cred', 'sgid'),
15117                               ('cred', 'user_ns')])},
15118               {'call': 'ioprio_set',
15119                'reason': set([('cred', 'user_ns'),
15120                               ('task_struct', 'cred')])},
15121               {'call': 'capset', 'reason': set([('cred', 'user_ns')])},
15122               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
15123               {'call': 'mq_timedreceive',
15124                'reason': set([('task_struct', 'cred')])},
15125               {'call': 'getresgid16', 'reason': set([('cred', 'user_ns')])},
15126               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
15127               {'call': 'sched_setaffinity',
15128                'reason': set([('cred', 'user_ns'),
15129                               ('task_struct', 'cred')])},
15130               {'call': 'setfsgid', 'reason': set([('cred', 'user_ns')])},
15131               {'call': 'unshare', 'reason': set([('cred', 'user_ns')])},
15132               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
15133               {'call': 'setreuid', 'reason': set([('cred', 'user_ns')])},
15134               {'call': 'semtimedop',
15135                'reason': set([('task_struct', 'cred')])},
15136               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
15137               {'call': 'sched_rr_get_interval',
15138                'reason': set([('task_struct', 'cred')])},
15139               {'call': 'epoll_create1',
15140                'reason': set([('cred', 'user_ns')])},
15141               {'call': 'getresuid', 'reason': set([('cred', 'user_ns')])},
15142               {'call': 'rt_sigprocmask',
15143                'reason': set([('task_struct', 'cred')])},
15144               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
15145               {'call': 'sigaltstack',
15146                'reason': set([('task_struct', 'cred')])},
15147               {'call': 'sched_setattr',
15148                'reason': set([('task_struct', 'cred')])},
15149               {'call': 'migrate_pages',
15150                'reason': set([('cred', 'user_ns'),
15151                               ('task_struct', 'cred')])},
15152               {'call': 'getitimer',
15153                'reason': set([('task_struct', 'cred')])},
15154               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
15155               {'call': 'setregid',
15156                'reason': set([('cred', 'egid'),
15157                               ('cred', 'gid'),
15158                               ('cred', 'sgid'),
15159                               ('cred', 'user_ns')])},
15160               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
15161               {'call': 'prlimit64',
15162                'reason': set([('cred', 'user_ns'),
15163                               ('task_struct', 'cred')])},
15164               {'call': 'perf_event_open',
15165                'reason': set([('task_struct', 'cred')])},
15166               {'call': 'getgroups16', 'reason': set([('cred', 'user_ns')])},
15167               {'call': 'rt_sigaction',
15168                'reason': set([('task_struct', 'cred')])},
15169               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
15170               {'call': 'getpriority',
15171                'reason': set([('cred', 'user_ns'),
15172                               ('task_struct', 'cred')])},
15173               {'call': 'sigaction',
```

```
15174                 'reason': set([('task_struct', 'cred')])},
15175                {'call': 'faccessat', 'reason': set([('cred', 'user_ns')])},
15176                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
15177                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
15178                {'call': 'get_robust_list',
15179                 'reason': set([('task_struct', 'cred')])},
15180                {'call': 'mq_timedsend',
15181                 'reason': set([('task_struct', 'cred')])},
15182                {'call': 'sched_getscheduler',
15183                 'reason': set([('task_struct', 'cred')])},
15184                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
15185                {'call': 'sched_getattr',
15186                 'reason': set([('task_struct', 'cred')])},
15187                {'call': 'getrusage',
15188                 'reason': set([('task_struct', 'cred')])},
15189                {'call': 'sched_setscheduler',
15190                 'reason': set([('task_struct', 'cred')])},
15191                {'call': 'setresuid', 'reason': set([('cred', 'user_ns')])},
15192                {'call': 'setitimer',
15193                 'reason': set([('task_struct', 'cred')])},
15194                {'call': 'ioprio_get',
15195                 'reason': set([('cred', 'user_ns'),
15196                                ('task_struct', 'cred')])},
15197                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
15198                {'call': 'setuid', 'reason': set([('cred', 'user_ns')])},
15199                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
15200                {'call': 'move_pages',
15201                 'reason': set([('task_struct', 'cred')])},
15202                {'call': 'getgroups', 'reason': set([('cred', 'user_ns')])},
15203                {'call': 'setpriority',
15204                 'reason': set([('cred', 'user_ns'),
15205                                ('task_struct', 'cred')])},
15206                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
15207                {'call': 'sched_getparam',
15208                 'reason': set([('task_struct', 'cred')])}],
15209  'setresuid': [{'call': 'keyctl',
15210                 'reason': set([('cred', 'user'),
15211                                ('cred', 'user_ns'),
15212                                ('task_struct', 'cred')])},
15213                {'call': 'rt_sigtimedwait',
15214                 'reason': set([('task_struct', 'cred')])},
15215                {'call': 'setfsuid',
15216                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15217                {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
15218                {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
15219                {'call': 'getresuid16',
15220                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15221                {'call': 'getresgid',
15222                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15223                {'call': 'sched_getaffinity',
15224                 'reason': set([('task_struct', 'cred')])},
15225                {'call': 'sched_setparam',
15226                 'reason': set([('task_struct', 'cred')])},
15227                {'call': 'setgid',
15228                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15229                {'call': 'ioprio_set',
15230                 'reason': set([('cred', 'user'),
15231                                ('cred', 'user_ns'),
15232                                ('task_struct', 'cred')])},
15233                {'call': 'capset',
15234                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15235                {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
15236                {'call': 'mq_timedreceive',
15237                 'reason': set([('task_struct', 'cred')])},
15238                {'call': 'getresgid16',
15239                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
```



```
15240                {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
15241                {'call': 'sched_setaffinity',
15242                 'reason': set([('cred', 'user'),
15243                                ('cred', 'user_ns'),
15244                                ('task_struct', 'cred')])},
15245                {'call': 'setfsgid',
15246                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15247                {'call': 'unshare',
15248                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15249                {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
15250                {'call': 'setreuid',
15251                 'reason': set([('cred', 'euid'),
15252                                ('cred', 'suid'),
15253                                ('cred', 'uid'),
15254                                ('cred', 'user'),
15255                                ('cred', 'user_ns')])},
15256                {'call': 'semtimedop',
15257                 'reason': set([('task_struct', 'cred')])},
15258                {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
15259                {'call': 'sched_rr_get_interval',
15260                 'reason': set([('task_struct', 'cred')])},
15261                {'call': 'epoll_create1',
15262                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15263                {'call': 'getresuid',
15264                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15265                {'call': 'rt_sigprocmask',
15266                 'reason': set([('task_struct', 'cred')])},
15267                {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
15268                {'call': 'sigaltstack',
15269                 'reason': set([('task_struct', 'cred')])},
15270                {'call': 'sched_setattr',
15271                 'reason': set([('task_struct', 'cred')])},
15272                {'call': 'migrate_pages',
15273                 'reason': set([('cred', 'user'),
15274                                ('cred', 'user_ns'),
15275                                ('task_struct', 'cred')])},
15276                {'call': 'getitimer',
15277                 'reason': set([('task_struct', 'cred')])},
15278                {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
15279                {'call': 'setresgid',
15280                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15281                {'call': 'setregid',
15282                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15283                {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
15284                {'call': 'prlimit64',
15285                 'reason': set([('cred', 'user'),
15286                                ('cred', 'user_ns'),
15287                                ('task_struct', 'cred')])},
15288                {'call': 'perf_event_open',
15289                 'reason': set([('task_struct', 'cred')])},
15290                {'call': 'getgroups16',
15291                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15292                {'call': 'rt_sigaction',
15293                 'reason': set([('task_struct', 'cred')])},
15294                {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
15295                {'call': 'getpriority',
15296                 'reason': set([('cred', 'user'),
15297                                ('cred', 'user_ns'),
15298                                ('task_struct', 'cred')])},
15299                {'call': 'sigaction',
15300                 'reason': set([('task_struct', 'cred')])},
15301                {'call': 'faccessat',
15302                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15303                {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
15304                {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
15305                {'call': 'get_robust_list',
```

```
15306                 'reason': set([('task_struct', 'cred')])},
15307                {'call': 'mq_timedsend',
15308                 'reason': set([('task_struct', 'cred')])},
15309                {'call': 'sched_getscheduler',
15310                 'reason': set([('task_struct', 'cred')])},
15311                {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
15312                {'call': 'sched_getattr',
15313                 'reason': set([('task_struct', 'cred')])},
15314                {'call': 'getrusage',
15315                 'reason': set([('task_struct', 'cred')])},
15316                {'call': 'sched_setscheduler',
15317                 'reason': set([('task_struct', 'cred')])},
15318                {'call': 'setitimer',
15319                 'reason': set([('task_struct', 'cred')])},
15320                {'call': 'ioprio_get',
15321                 'reason': set([('cred', 'user'),
15322                                ('cred', 'user_ns'),
15323                                ('task_struct', 'cred')])},
15324                {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
15325                {'call': 'setuid',
15326                 'reason': set([('cred', 'euid'),
15327                                ('cred', 'suid'),
15328                                ('cred', 'uid'),
15329                                ('cred', 'user'),
15330                                ('cred', 'user_ns')])},
15331                {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
15332                {'call': 'move_pages',
15333                 'reason': set([('task_struct', 'cred')])},
15334                {'call': 'getgroups',
15335                 'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15336                {'call': 'setpriority',
15337                 'reason': set([('cred', 'user'),
15338                                ('cred', 'user_ns'),
15339                                ('task_struct', 'cred')])},
15340                {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
15341                {'call': 'sched_getparam',
15342                 'reason': set([('task_struct', 'cred')])}],
15343  'setreuid': [{'call': 'keyctl',
15344                 'reason': set([('cred', 'user'),
15345                                ('cred', 'user_ns'),
15346                                ('task_struct', 'cred')])},
15347               {'call': 'rt_sigtimedwait',
15348                'reason': set([('task_struct', 'cred')])},
15349               {'call': 'setfsuid',
15350                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15351               {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
15352               {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
15353               {'call': 'getresuid16',
15354                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15355               {'call': 'getresgid',
15356                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15357               {'call': 'sched_getaffinity',
15358                'reason': set([('task_struct', 'cred')])},
15359               {'call': 'sched_setparam',
15360                'reason': set([('task_struct', 'cred')])},
15361               {'call': 'setgid',
15362                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15363               {'call': 'ioprio_set',
15364                'reason': set([('cred', 'user'),
15365                               ('cred', 'user_ns'),
15366                               ('task_struct', 'cred')])},
15367               {'call': 'capset',
15368                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15369               {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
15370               {'call': 'mq_timedreceive',
15371                'reason': set([('task_struct', 'cred')])},
```



```
15372               {'call': 'getresgid16',
15373                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15374               {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
15375               {'call': 'sched_setaffinity',
15376                'reason': set([('cred', 'user'),
15377                               ('cred', 'user_ns'),
15378                               ('task_struct', 'cred')])},
15379               {'call': 'setfsgid',
15380                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15381               {'call': 'unshare',
15382                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15383               {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
15384               {'call': 'semtimedop',
15385                'reason': set([('task_struct', 'cred')])},
15386               {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
15387               {'call': 'sched_rr_get_interval',
15388                'reason': set([('task_struct', 'cred')])},
15389               {'call': 'epoll_create1',
15390                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15391               {'call': 'getresuid',
15392                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15393               {'call': 'rt_sigprocmask',
15394                'reason': set([('task_struct', 'cred')])},
15395               {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
15396               {'call': 'sigaltstack',
15397                'reason': set([('task_struct', 'cred')])},
15398               {'call': 'sched_setattr',
15399                'reason': set([('task_struct', 'cred')])},
15400               {'call': 'migrate_pages',
15401                'reason': set([('cred', 'user'),
15402                               ('cred', 'user_ns'),
15403                               ('task_struct', 'cred')])},
15404               {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
15405               {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
15406               {'call': 'setresgid',
15407                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15408               {'call': 'setregid',
15409                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15410               {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
15411               {'call': 'prlimit64',
15412                'reason': set([('cred', 'user'),
15413                               ('cred', 'user_ns'),
15414                               ('task_struct', 'cred')])},
15415               {'call': 'perf_event_open',
15416                'reason': set([('task_struct', 'cred')])},
15417               {'call': 'getgroups16',
15418                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15419               {'call': 'rt_sigaction',
15420                'reason': set([('task_struct', 'cred')])},
15421               {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
15422               {'call': 'getpriority',
15423                'reason': set([('cred', 'user'),
15424                               ('cred', 'user_ns'),
15425                               ('task_struct', 'cred')])},
15426               {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
15427               {'call': 'faccessat',
15428                'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15429               {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
15430               {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
15431               {'call': 'get_robust_list',
15432                'reason': set([('task_struct', 'cred')])},
15433               {'call': 'mq_timedsend',
15434                'reason': set([('task_struct', 'cred')])},
15435               {'call': 'sched_getscheduler',
15436                'reason': set([('task_struct', 'cred')])},
15437               {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
```

```
15438              {'call': 'sched_getattr',
15439               'reason': set([('task_struct', 'cred')])},
15440              {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
15441              {'call': 'sched_setscheduler',
15442               'reason': set([('task_struct', 'cred')])},
15443              {'call': 'setresuid',
15444               'reason': set([('cred', 'euid'),
15445                              ('cred', 'suid'),
15446                              ('cred', 'uid'),
15447                              ('cred', 'user'),
15448                              ('cred', 'user_ns')])},
15449              {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
15450              {'call': 'ioprio_get',
15451               'reason': set([('cred', 'user'),
15452                              ('cred', 'user_ns'),
15453                              ('task_struct', 'cred')])},
15454              {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
15455              {'call': 'setuid',
15456               'reason': set([('cred', 'euid'),
15457                              ('cred', 'suid'),
15458                              ('cred', 'uid'),
15459                              ('cred', 'user'),
15460                              ('cred', 'user_ns')])},
15461              {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
15462              {'call': 'move_pages',
15463               'reason': set([('task_struct', 'cred')])},
15464              {'call': 'getgroups',
15465               'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15466              {'call': 'setpriority',
15467               'reason': set([('cred', 'user'),
15468                              ('cred', 'user_ns'),
15469                              ('task_struct', 'cred')])},
15470              {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
15471              {'call': 'sched_getparam',
15472               'reason': set([('task_struct', 'cred')])}],
15473  'setrlimit': [{'call': 'keyctl',
15474                 'reason': set([('task_struct', 'group_leader'),
15475                                ('task_struct', 'sighand')])},
15476                {'call': 'rt_sigtimedwait',
15477                 'reason': set([('task_struct', 'group_leader'),
15478                                ('task_struct', 'sighand')])},
15479                {'call': 'msgrcv',
15480                 'reason': set([('task_struct', 'group_leader'),
15481                                ('task_struct', 'sighand')])},
15482                {'call': 'kill',
15483                 'reason': set([('task_struct', 'group_leader'),
15484                                ('task_struct', 'sighand')])},
15485                {'call': 'sched_getaffinity',
15486                 'reason': set([('task_struct', 'group_leader'),
15487                                ('task_struct', 'sighand')])},
15488                {'call': 'sched_setparam',
15489                 'reason': set([('task_struct', 'group_leader'),
15490                                ('task_struct', 'sighand')])},
15491                {'call': 'ioprio_set',
15492                 'reason': set([('task_struct', 'group_leader'),
15493                                ('task_struct', 'sighand')])},
15494                {'call': 'getppid',
15495                 'reason': set([('task_struct', 'group_leader'),
15496                                ('task_struct', 'sighand')])},
15497                {'call': 'mq_timedreceive',
15498                 'reason': set([('task_struct', 'group_leader'),
15499                                ('task_struct', 'sighand')])},
15500                {'call': 'capget',
15501                 'reason': set([('task_struct', 'group_leader'),
15502                                ('task_struct', 'sighand')])},
15503                {'call': 'sched_setaffinity',
```



```
15504                 'reason': set([('task_struct', 'group_leader'),
15505                                ('task_struct', 'sighand')])},
15506              {'call': 'signal',
15507               'reason': set([('task_struct', 'group_leader'),
15508                              ('task_struct', 'sighand')])},
15509              {'call': 'semtimedop',
15510               'reason': set([('task_struct', 'group_leader'),
15511                              ('task_struct', 'sighand')])},
15512              {'call': 'umount',
15513               'reason': set([('task_struct', 'group_leader'),
15514                              ('task_struct', 'sighand')])},
15515              {'call': 'sched_rr_get_interval',
15516               'reason': set([('task_struct', 'group_leader'),
15517                              ('task_struct', 'sighand')])},
15518              {'call': 'rt_sigprocmask',
15519               'reason': set([('task_struct', 'group_leader'),
15520                              ('task_struct', 'sighand')])},
15521              {'call': 'setsid',
15522               'reason': set([('task_struct', 'group_leader'),
15523                              ('task_struct', 'sighand')])},
15524              {'call': 'sigaltstack',
15525               'reason': set([('task_struct', 'group_leader'),
15526                              ('task_struct', 'sighand')])},
15527              {'call': 'sched_setattr',
15528               'reason': set([('task_struct', 'group_leader'),
15529                              ('task_struct', 'sighand')])},
15530              {'call': 'migrate_pages',
15531               'reason': set([('task_struct', 'group_leader'),
15532                              ('task_struct', 'sighand')])},
15533              {'call': 'getitimer',
15534               'reason': set([('task_struct', 'group_leader'),
15535                              ('task_struct', 'sighand')])},
15536              {'call': 'setpgid',
15537               'reason': set([('task_struct', 'group_leader'),
15538                              ('task_struct', 'sighand')])},
15539              {'call': 'getsid',
15540               'reason': set([('task_struct', 'group_leader'),
15541                              ('task_struct', 'sighand')])},
15542              {'call': 'old_getrlimit',
15543               'reason': set([('rlimit', 'rlim_cur'),
15544                              ('rlimit', 'rlim_max')])},
15545              {'call': 'prlimit64',
15546               'reason': set([('rlimit', 'rlim_cur'),
15547                              ('rlimit', 'rlim_max'),
15548                              ('task_struct', 'group_leader'),
15549                              ('task_struct', 'sighand')])},
15550              {'call': 'perf_event_open',
15551               'reason': set([('task_struct', 'group_leader'),
15552                              ('task_struct', 'sighand')])},
15553              {'call': 'rt_sigaction',
15554               'reason': set([('task_struct', 'group_leader'),
15555                              ('task_struct', 'sighand')])},
15556              {'call': 'getpgid',
15557               'reason': set([('task_struct', 'group_leader'),
15558                              ('task_struct', 'sighand')])},
15559              {'call': 'getpriority',
15560               'reason': set([('task_struct', 'group_leader'),
15561                              ('task_struct', 'sighand')])},
15562              {'call': 'sigaction',
15563               'reason': set([('task_struct', 'group_leader'),
15564                              ('task_struct', 'sighand')])},
15565              {'call': 'setns',
15566               'reason': set([('task_struct', 'group_leader'),
15567                              ('task_struct', 'sighand')])},
15568              {'call': 'fork',
15569               'reason': set([('task_struct', 'group_leader'),
```

```
                                 ('task_struct', 'sighand')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'mq_timedsend',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'ptrace',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_getattr',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getrusage',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setitimer',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'ioprio_get',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'vfork',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'prctl',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'move_pages',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'setpriority',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'getrlimit',
                 'reason': set([('compat_rlimit', 'rlim_cur'),
                                ('compat_rlimit', 'rlim_max')])},
                {'call': 'clone',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])},
                {'call': 'sched_getparam',
                 'reason': set([('task_struct', 'group_leader'),
                                ('task_struct', 'sighand')])}],
 'setsid': [{'call': 'timer_create',
             'reason': set([('signal_struct', 'leader')])},
            {'call': 'exit_group',
             'reason': set([('signal_struct', 'leader')])}],
 'setsockopt': [{'call': 'accept4',
                 'reason': set([('proto_ops', 'compat_setsockopt')])}],
 'settimeofday': [{'call': 'waitid',
                   'reason': set([('timeval', 'tv_sec'),
                                  ('timeval', 'tv_usec')])},
                  {'call': 'adjtimex',
                   'reason': set([('timeval', 'tv_sec'),
                                  ('timeval', 'tv_usec')])},
                  {'call': 'getitimer',
                   'reason': set([('timeval', 'tv_sec'),
                                  ('timeval', 'tv_usec')])},
                  {'call': 'select',
                   'reason': set([('timeval', 'tv_sec'),
```



```
                                 ('timeval', 'tv_usec')])},
                {'call': 'wait4',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'getrusage',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'setitimer',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'clock_adjtime',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'alarm',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'ppoll',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])}],
 'setuid': [{'call': 'keyctl',
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setfsuid',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
            {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
            {'call': 'getresuid16',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'getresgid',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'setgid',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'ioprio_set',
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'capset',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'cred')])},
            {'call': 'getresgid16',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
            {'call': 'sched_setaffinity',
             'reason': set([('cred', 'user'),
                            ('cred', 'user_ns'),
                            ('task_struct', 'cred')])},
            {'call': 'setfsgid',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'unshare',
             'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
            {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
            {'call': 'setreuid',
             'reason': set([('cred', 'suid'),
                            ('cred', 'uid'),
                            ('cred', 'user'),
                            ('cred', 'user_ns')])},
            {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
```

```
15702             {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
15703             {'call': 'sched_rr_get_interval',
15704              'reason': set([('task_struct', 'cred')])},
15705             {'call': 'epoll_create1',
15706              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15707             {'call': 'getresuid',
15708              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15709             {'call': 'rt_sigprocmask',
15710              'reason': set([('task_struct', 'cred')])},
15711             {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
15712             {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
15713             {'call': 'sched_setattr',
15714              'reason': set([('task_struct', 'cred')])},
15715             {'call': 'migrate_pages',
15716              'reason': set([('cred', 'user'),
15717                             ('cred', 'user_ns'),
15718                             ('task_struct', 'cred')])},
15719             {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
15720             {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
15721             {'call': 'setresgid',
15722              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15723             {'call': 'setregid',
15724              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15725             {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
15726             {'call': 'prlimit64',
15727              'reason': set([('cred', 'user'),
15728                             ('cred', 'user_ns'),
15729                             ('task_struct', 'cred')])},
15730             {'call': 'perf_event_open',
15731              'reason': set([('task_struct', 'cred')])},
15732             {'call': 'getgroups16',
15733              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15734             {'call': 'rt_sigaction',
15735              'reason': set([('task_struct', 'cred')])},
15736             {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
15737             {'call': 'getpriority',
15738              'reason': set([('cred', 'user'),
15739                             ('cred', 'user_ns'),
15740                             ('task_struct', 'cred')])},
15741             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
15742             {'call': 'faccessat',
15743              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15744             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
15745             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
15746             {'call': 'get_robust_list',
15747              'reason': set([('task_struct', 'cred')])},
15748             {'call': 'mq_timedsend',
15749              'reason': set([('task_struct', 'cred')])},
15750             {'call': 'sched_getscheduler',
15751              'reason': set([('task_struct', 'cred')])},
15752             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
15753             {'call': 'sched_getattr',
15754              'reason': set([('task_struct', 'cred')])},
15755             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
15756             {'call': 'sched_setscheduler',
15757              'reason': set([('task_struct', 'cred')])},
15758             {'call': 'setresuid',
15759              'reason': set([('cred', 'suid'),
15760                             ('cred', 'uid'),
15761                             ('cred', 'user'),
15762                             ('cred', 'user_ns')])},
15763             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
15764             {'call': 'ioprio_get',
15765              'reason': set([('cred', 'user'),
15766                             ('cred', 'user_ns'),
15767                             ('task_struct', 'cred')])},
```



```
15768             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
15769             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
15770             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
15771             {'call': 'getgroups',
15772              'reason': set([('cred', 'user'), ('cred', 'user_ns')])},
15773             {'call': 'setpriority',
15774              'reason': set([('cred', 'user'),
15775                             ('cred', 'user_ns'),
15776                             ('task_struct', 'cred')])},
15777             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
15778             {'call': 'sched_getparam',
15779              'reason': set([('task_struct', 'cred')])}],
15780  'setxattr': [{'call': 'eventfd2',
15781                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15782               {'call': 'swapoff',
15783                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15784               {'call': 'pivot_root',
15785                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15786               {'call': 'memfd_create',
15787                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15788               {'call': 'remap_file_pages',
15789                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15790               {'call': 'dup3',
15791                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15792               {'call': 'unshare',
15793                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15794               {'call': 'epoll_create1',
15795                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15796               {'call': 'epoll_ctl',
15797                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15798               {'call': 'flock',
15799                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15800               {'call': 'openat',
15801                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15802               {'call': 'lookup_dcookie',
15803                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15804               {'call': 'uselib',
15805                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15806               {'call': 'accept4',
15807                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15808               {'call': 'socketpair',
15809                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15810               {'call': 'getcwd',
15811                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15812               {'call': 'shmat',
15813                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15814               {'call': 'socket',
15815                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15816               {'call': 'pipe2',
15817                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15818               {'call': 'perf_event_open',
15819                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15820               {'call': 'shmdt',
15821                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15822               {'call': 'quotactl',
15823                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15824               {'call': 'acct',
15825                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15826               {'call': 'open',
15827                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15828               {'call': 'dup',
15829                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15830               {'call': 'setns',
15831                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15832               {'call': 'shmctl',
15833                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```

```
15834               {'call': 'swapon',
15835                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15836               {'call': 'mmap_pgoff',
15837                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15838               {'call': 'mq_open',
15839                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
15840               {'call': 'open_by_handle_at',
15841                'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
15842  'shmat': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
15843            {'call': 'rt_sigtimedwait',
15844             'reason': set([('task_struct', 'mm')])},
15845            {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
15846            {'call': 'eventfd2', 'reason': set([('path', 'dentry')])},
15847            {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
15848            {'call': 'swapoff', 'reason': set([('path', 'dentry')])},
15849            {'call': 'sched_getaffinity',
15850             'reason': set([('task_struct', 'mm')])},
15851            {'call': 'sched_setparam', 'reason': set([('task_struct', 'mm')])},
15852            {'call': 'pivot_root', 'reason': set([('path', 'dentry')])},
15853            {'call': 'memfd_create', 'reason': set([('path', 'dentry')])},
15854            {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
15855            {'call': 'remap_file_pages', 'reason': set([('path', 'dentry')])},
15856            {'call': 'dup3', 'reason': set([('path', 'dentry')])},
15857            {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
15858            {'call': 'mq_timedreceive',
15859             'reason': set([('task_struct', 'mm')])},
15860            {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
15861            {'call': 'sched_setaffinity',
15862             'reason': set([('task_struct', 'mm')])},
15863            {'call': 'unshare', 'reason': set([('path', 'dentry')])},
15864            {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
15865            {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
15866            {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
15867            {'call': 'sched_rr_get_interval',
15868             'reason': set([('task_struct', 'mm')])},
15869            {'call': 'epoll_create1', 'reason': set([('path', 'dentry')])},
15870            {'call': 'epoll_ctl', 'reason': set([('path', 'dentry')])},
15871            {'call': 'flock', 'reason': set([('path', 'dentry')])},
15872            {'call': 'openat', 'reason': set([('path', 'dentry')])},
15873            {'call': 'lookup_dcookie', 'reason': set([('path', 'dentry')])},
15874            {'call': 'uselib', 'reason': set([('path', 'dentry')])},
15875            {'call': 'rt_sigprocmask', 'reason': set([('task_struct', 'mm')])},
15876            {'call': 'accept4', 'reason': set([('path', 'dentry')])},
15877            {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
15878            {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
15879            {'call': 'sched_setattr', 'reason': set([('task_struct', 'mm')])},
15880            {'call': 'socketpair', 'reason': set([('path', 'dentry')])},
15881            {'call': 'migrate_pages', 'reason': set([('task_struct', 'mm')])},
15882            {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
15883            {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
15884            {'call': 'getcwd', 'reason': set([('path', 'dentry')])},
15885            {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
15886            {'call': 'socket', 'reason': set([('path', 'dentry')])},
15887            {'call': 'pipe2', 'reason': set([('path', 'dentry')])},
15888            {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
15889            {'call': 'perf_event_open',
15890             'reason': set([('path', 'dentry'), ('task_struct', 'mm')])},
15891            {'call': 'shmdt', 'reason': set([('path', 'dentry')])},
15892            {'call': 'quotactl', 'reason': set([('path', 'dentry')])},
15893            {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
15894            {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
15895            {'call': 'acct', 'reason': set([('path', 'dentry')])},
15896            {'call': 'open', 'reason': set([('path', 'dentry')])},
15897            {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
15898            {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
15899            {'call': 'dup', 'reason': set([('path', 'dentry')])},
```



```
15900            {'call': 'setns',
15901             'reason': set([('path', 'dentry'), ('task_struct', 'mm')])},
15902            {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
15903            {'call': 'get_robust_list',
15904             'reason': set([('task_struct', 'mm')])},
15905            {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
15906            {'call': 'sched_getscheduler',
15907             'reason': set([('task_struct', 'mm')])},
15908            {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
15909            {'call': 'shmctl',
15910             'reason': set([('path', 'dentry'),
15911                            ('shmid_kernel', 'shm_file')])},
15912            {'call': 'swapon', 'reason': set([('path', 'dentry')])},
15913            {'call': 'sched_getattr', 'reason': set([('task_struct', 'mm')])},
15914            {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
15915            {'call': 'sched_setscheduler',
15916             'reason': set([('task_struct', 'mm')])},
15917            {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
15918            {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
15919            {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
15920            {'call': 'mmap_pgoff', 'reason': set([('path', 'dentry')])},
15921            {'call': 'prctl', 'reason': set([('task_struct', 'mm')])},
15922            {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
15923            {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
15924            {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
15925            {'call': 'mq_open', 'reason': set([('path', 'dentry')])},
15926            {'call': 'sched_getparam', 'reason': set([('task_struct', 'mm')])},
15927            {'call': 'open_by_handle_at', 'reason': set([('path', 'dentry')])}],
15928  'shmctl': [{'call': 'keyctl',
15929              'reason': set([('mm_segment_t', 'seg'),
15930                             ('task_struct', 'cred')])},
15931             {'call': 'rt_sigtimedwait',
15932              'reason': set([('mm_segment_t', 'seg'),
15933                             ('task_struct', 'cred')])},
15934             {'call': 'msgrcv',
15935              'reason': set([('ipc_namespace', 'user_ns'),
15936                             ('kern_ipc_perm', 'deleted'),
15937                             ('kern_ipc_perm', 'mode'),
15938                             ('mm_segment_t', 'seg'),
15939                             ('task_struct', 'cred')])},
15940             {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
15941             {'call': 'mq_unlink',
15942              'reason': set([('ipc_namespace', 'user_ns')])},
15943             {'call': 'kill',
15944              'reason': set([('mm_segment_t', 'seg'),
15945                             ('task_struct', 'cred')])},
15946             {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
15947             {'call': 'msgget', 'reason': set([('ipc_namespace', 'user_ns')])},
15948             {'call': 'sched_getaffinity',
15949              'reason': set([('mm_segment_t', 'seg'),
15950                             ('task_struct', 'cred')])},
15951             {'call': 'sched_setparam',
15952              'reason': set([('mm_segment_t', 'seg'),
15953                             ('task_struct', 'cred')])},
15954             {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
15955             {'call': 'ioprio_set',
15956              'reason': set([('mm_segment_t', 'seg'),
15957                             ('task_struct', 'cred')])},
15958             {'call': 'remap_file_pages', 'reason': set([('file', 'f_op')])},
15959             {'call': 'dup3', 'reason': set([('file', 'f_op')])},
15960             {'call': 'getppid',
15961              'reason': set([('mm_segment_t', 'seg'),
15962                             ('task_struct', 'cred')])},
15963             {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
15964             {'call': 'mq_timedreceive',
15965              'reason': set([('mm_segment_t', 'seg'),
```

```
                                ('task_struct', 'cred')])},
            {'call': 'capget',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'sched_setaffinity',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'signal',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'semtimedop',
             'reason': set([('ipc_namespace', 'user_ns'),
                            ('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode'),
                            ('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'umount',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
            {'call': 'semctl',
             'reason': set([('ipc_namespace', 'user_ns'),
                            ('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'epoll_ctl', 'reason': set([('file', 'f_op')])},
            {'call': 'flock', 'reason': set([('file', 'f_op')])},
            {'call': 'openat', 'reason': set([('file', 'f_op')])},
            {'call': 'shmget', 'reason': set([('ipc_namespace', 'user_ns')])},
            {'call': 'uselib', 'reason': set([('file', 'f_op')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'accept4', 'reason': set([('file', 'f_op')])},
            {'call': 'msgctl',
             'reason': set([('ipc_namespace', 'user_ns'),
                            ('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'setsid',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'sigaltstack',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'sched_setattr',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
            {'call': 'migrate_pages',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'getitimer',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'setpgid',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'semget', 'reason': set([('ipc_namespace', 'user_ns')])},
            {'call': 'getsid',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'shmat',
             'reason': set([('file', 'f_op'),
                            ('ipc_namespace', 'user_ns'),
```



```
                            ('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'socket', 'reason': set([('file', 'f_op')])},
            {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
            {'call': 'prlimit64',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'perf_event_open',
             'reason': set([('file', 'f_op'),
                            ('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
            {'call': 'rt_sigaction',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'getpgid',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'acct', 'reason': set([('file', 'f_op')])},
            {'call': 'open', 'reason': set([('file', 'f_op')])},
            {'call': 'getpriority',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'sigaction',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'dup', 'reason': set([('file', 'f_op')])},
            {'call': 'setns',
             'reason': set([('file', 'f_op'),
                            ('ipc_namespace', 'user_ns'),
                            ('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'fork',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'get_robust_list',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'mq_timedsend',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'sched_getscheduler',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'ptrace',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'swapon', 'reason': set([('file', 'f_op')])},
            {'call': 'sched_getattr',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'getrusage',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'sched_setscheduler',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'setitimer',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'ioprio_get',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'vfork',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
```

```
            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
            {'call': 'prctl',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'move_pages',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'msgsnd',
             'reason': set([('ipc_namespace', 'user_ns'),
                            ('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'setpriority',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'clone',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'mq_open',
             'reason': set([('file', 'f_op'), ('ipc_namespace', 'user_ns')])},
            {'call': 'sched_getparam',
             'reason': set([('mm_segment_t', 'seg'),
                            ('task_struct', 'cred')])},
            {'call': 'open_by_handle_at', 'reason': set([('file', 'f_op')])}],
 'shmdt': [{'call': 'remap_file_pages',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'brk',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'get_mempolicy',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'munlockall',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'pkey_mprotect',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'madvise',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'mprotect',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'mremap',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'prctl',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'munlock',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
                           ('vm_area_struct', 'vm_pgoff')])},
           {'call': 'mincore',
            'reason': set([('vm_area_struct', 'vm_file'),
                           ('vm_area_struct', 'vm_ops'),
```



```
                           ('vm_area_struct', 'vm_pgoff')])},
          {'call': 'mlockall',
           'reason': set([('vm_area_struct', 'vm_file'),
                          ('vm_area_struct', 'vm_ops'),
                          ('vm_area_struct', 'vm_pgoff')])}],
 'sigaction': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setparam',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioprio_set',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedreceive',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'semtimedop',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_rr_get_interval',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigprocmask',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sigaltstack',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setattr',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'migrate_pages',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getitimer',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'prlimit64',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'perf_event_open',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigaction',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getpriority',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'get_robust_list',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedsend',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getscheduler',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getattr',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getrusage',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setscheduler',
                'reason': set([('mm_segment_t', 'seg')])},
```

```
16230                {'call': 'setitimer',
16231                 'reason': set([('mm_segment_t', 'seg')])},
16232                {'call': 'ioprio_get',
16233                 'reason': set([('mm_segment_t', 'seg')])},
16234                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
16235                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
16236                {'call': 'move_pages',
16237                 'reason': set([('mm_segment_t', 'seg')])},
16238                {'call': 'setpriority',
16239                 'reason': set([('mm_segment_t', 'seg')])},
16240                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
16241                {'call': 'sched_getparam',
16242                 'reason': set([('mm_segment_t', 'seg')])}],
16243  'signalfd4': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
16244                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
16245                {'call': 'eventfd2', 'reason': set([('file', 'f_op')])},
16246                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
16247                {'call': 'swapoff', 'reason': set([('file', 'f_op')])},
16248                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
16249                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
16250                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
16251                {'call': 'writev', 'reason': set([('fd', 'flags')])},
16252                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
16253                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
16254                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
16255                {'call': 'memfd_create', 'reason': set([('file', 'f_op')])},
16256                {'call': 'remap_file_pages',
16257                 'reason': set([('file', 'f_op')])},
16258                {'call': 'dup3', 'reason': set([('file', 'f_op')])},
16259                {'call': 'read', 'reason': set([('fd', 'flags')])},
16260                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
16261                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
16262                {'call': 'utime', 'reason': set([('fd', 'flags')])},
16263                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
16264                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
16265                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
16266                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
16267                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
16268                {'call': 'epoll_create1', 'reason': set([('file', 'f_op')])},
16269                {'call': 'tee', 'reason': set([('fd', 'flags')])},
16270                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
16271                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
16272                {'call': 'connect', 'reason': set([('fd', 'flags')])},
16273                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
16274                {'call': 'epoll_ctl',
16275                 'reason': set([('fd', 'flags'), ('file', 'f_op')])},
16276                {'call': 'flock',
16277                 'reason': set([('fd', 'flags'), ('file', 'f_op')])},
16278                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
16279                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
16280                {'call': 'openat', 'reason': set([('file', 'f_op')])},
16281                {'call': 'uselib', 'reason': set([('file', 'f_op')])},
16282                {'call': 'accept4',
16283                 'reason': set([('fd', 'flags'), ('file', 'f_op')])},
16284                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
16285                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
16286                {'call': 'socketpair', 'reason': set([('file', 'f_op')])},
16287                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
16288                {'call': 'inotify_add_watch',
16289                 'reason': set([('fd', 'flags')])},
16290                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
16291                {'call': 'splice', 'reason': set([('fd', 'flags')])},
16292                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
16293                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
16294                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
16295                {'call': 'shmat', 'reason': set([('file', 'f_op')])},
```



```
16296                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
16297                {'call': 'socket', 'reason': set([('file', 'f_op')])},
16298                {'call': 'pipe2', 'reason': set([('file', 'f_op')])},
16299                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
16300                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
16301                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
16302                {'call': 'perf_event_open',
16303                 'reason': set([('fd', 'flags'), ('file', 'f_op')])},
16304                {'call': 'shmdt', 'reason': set([('file', 'f_op')])},
16305                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
16306                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
16307                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
16308                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
16309                {'call': 'acct', 'reason': set([('file', 'f_op')])},
16310                {'call': 'open', 'reason': set([('file', 'f_op')])},
16311                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
16312                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
16313                {'call': 'dup', 'reason': set([('file', 'f_op')])},
16314                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
16315                {'call': 'setns', 'reason': set([('file', 'f_op')])},
16316                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
16317                {'call': 'listen', 'reason': set([('fd', 'flags')])},
16318                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
16319                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
16320                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
16321                {'call': 'shmctl', 'reason': set([('file', 'f_op')])},
16322                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
16323                {'call': 'swapon', 'reason': set([('file', 'f_op')])},
16324                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
16325                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
16326                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
16327                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_op')])},
16328                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
16329                {'call': 'readv', 'reason': set([('fd', 'flags')])},
16330                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
16331                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
16332                {'call': 'write', 'reason': set([('fd', 'flags')])},
16333                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
16334                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
16335                {'call': 'mq_open', 'reason': set([('file', 'f_op')])},
16336                {'call': 'open_by_handle_at',
16337                 'reason': set([('file', 'f_op')])},
16338                {'call': 'bind', 'reason': set([('fd', 'flags')])},
16339                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
16340                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
16341  'splice': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
16342             {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
16343             {'call': 'eventfd2',
16344              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16345             {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
16346             {'call': 'swapoff',
16347              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16348             {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
16349             {'call': 'readahead', 'reason': set([('fd', 'flags')])},
16350             {'call': 'getdents', 'reason': set([('fd', 'flags')])},
16351             {'call': 'writev', 'reason': set([('fd', 'flags')])},
16352             {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
16353             {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
16354             {'call': 'pread64', 'reason': set([('fd', 'flags')])},
16355             {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
16356             {'call': 'memfd_create',
16357              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16358             {'call': 'remap_file_pages',
16359              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16360             {'call': 'dup3',
16361              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
```

```
16362             {'call': 'read', 'reason': set([('fd', 'flags')])},
16363             {'call': 'fchown', 'reason': set([('fd', 'flags')])},
16364             {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
16365             {'call': 'utime', 'reason': set([('fd', 'flags')])},
16366             {'call': 'fsync', 'reason': set([('fd', 'flags')])},
16367             {'call': 'bpf', 'reason': set([('fd', 'flags')])},
16368             {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
16369             {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
16370             {'call': 'sendto', 'reason': set([('fd', 'flags')])},
16371             {'call': 'epoll_create1',
16372              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16373             {'call': 'tee', 'reason': set([('fd', 'flags')])},
16374             {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
16375             {'call': 'lseek', 'reason': set([('fd', 'flags')])},
16376             {'call': 'connect', 'reason': set([('fd', 'flags')])},
16377             {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
16378             {'call': 'epoll_ctl',
16379              'reason': set([('fd', 'flags'),
16380                             ('file', 'f_flags'),
16381                             ('file', 'f_mode')])},
16382             {'call': 'flock',
16383              'reason': set([('fd', 'flags'),
16384                             ('file', 'f_flags'),
16385                             ('file', 'f_mode')])},
16386             {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
16387             {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
16388             {'call': 'openat',
16389              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16390             {'call': 'uselib',
16391              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16392             {'call': 'accept4',
16393              'reason': set([('fd', 'flags'),
16394                             ('file', 'f_flags'),
16395                             ('file', 'f_mode')])},
16396             {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
16397             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
16398             {'call': 'socketpair',
16399              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16400             {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
16401             {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
16402             {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
16403             {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
16404             {'call': 'preadv', 'reason': set([('fd', 'flags')])},
16405             {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
16406             {'call': 'shmat',
16407              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16408             {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
16409             {'call': 'socket',
16410              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16411             {'call': 'pipe2',
16412              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16413             {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
16414             {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
16415             {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
16416             {'call': 'perf_event_open',
16417              'reason': set([('fd', 'flags'),
16418                             ('file', 'f_flags'),
16419                             ('file', 'f_mode')])},
16420             {'call': 'shmdt',
16421              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16422             {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
16423             {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
16424             {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
16425             {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
16426             {'call': 'acct',
16427              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
```



```
16428             {'call': 'open',
16429              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16430             {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
16431             {'call': 'mq_getsetattr',
16432              'reason': set([('fd', 'flags'), ('file', 'f_flags')])},
16433             {'call': 'dup',
16434              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16435             {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
16436             {'call': 'setns',
16437              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16438             {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
16439             {'call': 'listen', 'reason': set([('fd', 'flags')])},
16440             {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
16441             {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
16442             {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
16443             {'call': 'shmctl',
16444              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16445             {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
16446             {'call': 'swapon',
16447              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16448             {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
16449             {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
16450             {'call': 'llseek', 'reason': set([('fd', 'flags')])},
16451             {'call': 'mmap_pgoff',
16452              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16453             {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
16454             {'call': 'readv', 'reason': set([('fd', 'flags')])},
16455             {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
16456             {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
16457             {'call': 'write', 'reason': set([('fd', 'flags')])},
16458             {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
16459             {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
16460             {'call': 'mq_open',
16461              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16462             {'call': 'open_by_handle_at',
16463              'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
16464             {'call': 'bind', 'reason': set([('fd', 'flags')])},
16465             {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
16466             {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
16467  'stat': [{'call': 'lstat',
16468            'reason': set([('__old_kernel_stat', 'st_ino'),
16469                           ('__old_kernel_stat', 'st_nlink')])},
16470           {'call': 'fstat',
16471            'reason': set([('__old_kernel_stat', 'st_ino'),
16472                           ('__old_kernel_stat', 'st_nlink'),
16473                           ('kstat', 'dev'),
16474                           ('kstat', 'ino'),
16475                           ('kstat', 'nlink'),
16476                           ('kstat', 'rdev')])},
16477           {'call': 'newfstat',
16478            'reason': set([('kstat', 'dev'),
16479                           ('kstat', 'ino'),
16480                           ('kstat', 'nlink'),
16481                           ('kstat', 'rdev')])}],
16482  'statfs': [{'call': 'ustat',
16483              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
16484             {'call': 'fstatfs',
16485              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
16486             {'call': 'fstatfs64',
16487              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
16488             {'call': 'statfs64',
16489              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])}],
16490  'statfs64': [{'call': 'ustat',
16491                'reason': set([('kstatfs', 'f_ffree'),
16492                               ('kstatfs', 'f_files')])},
16493               {'call': 'fstatfs',
```

```
16494                'reason': set([('kstatfs', 'f_ffree'),
16495                               ('kstatfs', 'f_files')])},
16496              {'call': 'statfs',
16497               'reason': set([('kstatfs', 'f_ffree'),
16498                              ('kstatfs', 'f_files')])},
16499              {'call': 'fstatfs64',
16500               'reason': set([('kstatfs', 'f_ffree'),
16501                              ('kstatfs', 'f_files')])}],
16502  'swapoff': [{'call': 'keyctl', 'reason': set([('task_struct', 'mm')])},
16503              {'call': 'rt_sigtimedwait',
16504               'reason': set([('task_struct', 'mm')])},
16505              {'call': 'msgrcv', 'reason': set([('task_struct', 'mm')])},
16506              {'call': 'eventfd2', 'reason': set([('file', 'f_mapping')])},
16507              {'call': 'kill', 'reason': set([('task_struct', 'mm')])},
16508              {'call': 'sched_getaffinity',
16509               'reason': set([('task_struct', 'mm')])},
16510              {'call': 'sched_setparam',
16511               'reason': set([('task_struct', 'mm')])},
16512              {'call': 'memfd_create', 'reason': set([('file', 'f_mapping')])},
16513              {'call': 'ioprio_set', 'reason': set([('task_struct', 'mm')])},
16514              {'call': 'remap_file_pages',
16515               'reason': set([('file', 'f_mapping')])},
16516              {'call': 'dup3', 'reason': set([('file', 'f_mapping')])},
16517              {'call': 'getppid', 'reason': set([('task_struct', 'mm')])},
16518              {'call': 'mq_timedreceive',
16519               'reason': set([('task_struct', 'mm')])},
16520              {'call': 'capget', 'reason': set([('task_struct', 'mm')])},
16521              {'call': 'sched_setaffinity',
16522               'reason': set([('task_struct', 'mm')])},
16523              {'call': 'signal', 'reason': set([('task_struct', 'mm')])},
16524              {'call': 'semtimedop', 'reason': set([('task_struct', 'mm')])},
16525              {'call': 'umount', 'reason': set([('task_struct', 'mm')])},
16526              {'call': 'sched_rr_get_interval',
16527               'reason': set([('task_struct', 'mm')])},
16528              {'call': 'epoll_create1',
16529               'reason': set([('file', 'f_mapping')])},
16530              {'call': 'epoll_ctl', 'reason': set([('file', 'f_mapping')])},
16531              {'call': 'flock', 'reason': set([('file', 'f_mapping')])},
16532              {'call': 'openat', 'reason': set([('file', 'f_mapping')])},
16533              {'call': 'uselib', 'reason': set([('file', 'f_mapping')])},
16534              {'call': 'rt_sigprocmask',
16535               'reason': set([('task_struct', 'mm')])},
16536              {'call': 'accept4', 'reason': set([('file', 'f_mapping')])},
16537              {'call': 'setsid', 'reason': set([('task_struct', 'mm')])},
16538              {'call': 'sigaltstack', 'reason': set([('task_struct', 'mm')])},
16539              {'call': 'sched_setattr',
16540               'reason': set([('task_struct', 'mm')])},
16541              {'call': 'socketpair', 'reason': set([('file', 'f_mapping')])},
16542              {'call': 'migrate_pages',
16543               'reason': set([('task_struct', 'mm')])},
16544              {'call': 'getitimer', 'reason': set([('task_struct', 'mm')])},
16545              {'call': 'setpgid', 'reason': set([('task_struct', 'mm')])},
16546              {'call': 'getsid', 'reason': set([('task_struct', 'mm')])},
16547              {'call': 'shmat', 'reason': set([('file', 'f_mapping')])},
16548              {'call': 'socket', 'reason': set([('file', 'f_mapping')])},
16549              {'call': 'pipe2', 'reason': set([('file', 'f_mapping')])},
16550              {'call': 'prlimit64', 'reason': set([('task_struct', 'mm')])},
16551              {'call': 'perf_event_open',
16552               'reason': set([('file', 'f_mapping'), ('task_struct', 'mm')])},
16553              {'call': 'shmdt', 'reason': set([('file', 'f_mapping')])},
16554              {'call': 'rt_sigaction', 'reason': set([('task_struct', 'mm')])},
16555              {'call': 'getpgid', 'reason': set([('task_struct', 'mm')])},
16556              {'call': 'acct', 'reason': set([('file', 'f_mapping')])},
16557              {'call': 'open', 'reason': set([('file', 'f_mapping')])},
16558              {'call': 'getpriority', 'reason': set([('task_struct', 'mm')])},
16559              {'call': 'sigaction', 'reason': set([('task_struct', 'mm')])},
```



```
16560              {'call': 'dup', 'reason': set([('file', 'f_mapping')])},
16561              {'call': 'setns',
16562               'reason': set([('file', 'f_mapping'), ('task_struct', 'mm')])},
16563              {'call': 'fork', 'reason': set([('task_struct', 'mm')])},
16564              {'call': 'get_robust_list',
16565               'reason': set([('task_struct', 'mm')])},
16566              {'call': 'mq_timedsend', 'reason': set([('task_struct', 'mm')])},
16567              {'call': 'sched_getscheduler',
16568               'reason': set([('task_struct', 'mm')])},
16569              {'call': 'ptrace', 'reason': set([('task_struct', 'mm')])},
16570              {'call': 'shmctl', 'reason': set([('file', 'f_mapping')])},
16571              {'call': 'swapon',
16572               'reason': set([('file', 'f_mapping'),
16573                              ('page', 'private'),
16574                              ('swap_info_struct', 'cluster_info'),
16575                              ('swap_info_struct', 'flags'),
16576                              ('swap_info_struct', 'inuse_pages'),
16577                              ('swap_info_struct', 'pages'),
16578                              ('swap_info_struct', 'prio'),
16579                              ('swap_info_struct', 'swap_map')])},
16580              {'call': 'sched_getattr',
16581               'reason': set([('task_struct', 'mm')])},
16582              {'call': 'getrusage', 'reason': set([('task_struct', 'mm')])},
16583              {'call': 'sched_setscheduler',
16584               'reason': set([('task_struct', 'mm')])},
16585              {'call': 'setitimer', 'reason': set([('task_struct', 'mm')])},
16586              {'call': 'ioprio_get', 'reason': set([('task_struct', 'mm')])},
16587              {'call': 'vfork', 'reason': set([('task_struct', 'mm')])},
16588              {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mapping')])},
16589              {'call': 'prctl', 'reason': set([('task_struct', 'mm')])},
16590              {'call': 'move_pages', 'reason': set([('task_struct', 'mm')])},
16591              {'call': 'setpriority', 'reason': set([('task_struct', 'mm')])},
16592              {'call': 'clone', 'reason': set([('task_struct', 'mm')])},
16593              {'call': 'mq_open', 'reason': set([('file', 'f_mapping')])},
16594              {'call': 'sched_getparam',
16595               'reason': set([('task_struct', 'mm')])},
16596              {'call': 'open_by_handle_at',
16597               'reason': set([('file', 'f_mapping')])}],
16598  'swapon': [{'call': 'mq_unlink', 'reason': set([('inode', 'i_flags')])},
16599             {'call': 'swapoff',
16600              'reason': set([('inode', 'i_flags'),
16601                             ('swap_info_struct', 'bdev'),
16602                             ('swap_info_struct', 'flags'),
16603                             ('swap_info_struct', 'percpu_cluster'),
16604                             ('swap_info_struct', 'type')])},
16605             {'call': 'fchmod', 'reason': set([('inode', 'i_flags')])},
16606             {'call': 'memfd_create', 'reason': set([('inode', 'i_flags')])},
16607             {'call': 'readlinkat', 'reason': set([('inode', 'i_flags')])},
16608             {'call': 'fchown', 'reason': set([('inode', 'i_flags')])},
16609             {'call': 'mq_timedreceive',
16610              'reason': set([('inode', 'i_flags')])},
16611             {'call': 'uselib', 'reason': set([('inode', 'i_flags')])},
16612             {'call': 'fchmodat', 'reason': set([('inode', 'i_flags')])},
16613             {'call': 'inotify_add_watch',
16614              'reason': set([('inode', 'i_flags')])},
16615             {'call': 'ftruncate', 'reason': set([('inode', 'i_flags')])},
16616             {'call': 'ioctl', 'reason': set([('inode', 'i_flags')])},
16617             {'call': 'linkat', 'reason': set([('inode', 'i_flags')])},
16618             {'call': 'unlink', 'reason': set([('inode', 'i_flags')])},
16619             {'call': 'mq_getsetattr', 'reason': set([('inode', 'i_flags')])},
16620             {'call': 'faccessat', 'reason': set([('inode', 'i_flags')])},
16621             {'call': 'mq_timedsend', 'reason': set([('inode', 'i_flags')])},
16622             {'call': 'fchownat', 'reason': set([('inode', 'i_flags')])},
16623             {'call': 'mq_notify', 'reason': set([('inode', 'i_flags')])},
16624             {'call': 'sendfile', 'reason': set([('inode', 'i_flags')])},
16625             {'call': 'unlinkat', 'reason': set([('inode', 'i_flags')])},
```

```
16626             {'call': 'sendfile64', 'reason': set([('inode', 'i_flags')])}],
16627  'symlinkat': [{'call': 'sysfs',
16628                 'reason': set([('filename', 'name'),
16629                                ('filename', 'refcnt')])},
16630                {'call': 'mq_unlink',
16631                 'reason': set([('filename', 'name'),
16632                                ('filename', 'refcnt')])},
16633                {'call': 'swapoff',
16634                 'reason': set([('filename', 'name'),
16635                                ('filename', 'refcnt')])},
16636                {'call': 'openat',
16637                 'reason': set([('filename', 'name'),
16638                                ('filename', 'refcnt')])},
16639                {'call': 'uselib',
16640                 'reason': set([('filename', 'name'),
16641                                ('filename', 'refcnt')])},
16642                {'call': 'renameat2',
16643                 'reason': set([('filename', 'name'),
16644                                ('filename', 'refcnt')])},
16645                {'call': 'quotactl',
16646                 'reason': set([('filename', 'name'),
16647                                ('filename', 'refcnt')])},
16648                {'call': 'acct',
16649                 'reason': set([('filename', 'name'),
16650                                ('filename', 'refcnt')])},
16651                {'call': 'open',
16652                 'reason': set([('filename', 'name'),
16653                                ('filename', 'refcnt')])},
16654                {'call': 'unlink',
16655                 'reason': set([('filename', 'name'),
16656                                ('filename', 'refcnt')])},
16657                {'call': 'rmdir',
16658                 'reason': set([('filename', 'name'),
16659                                ('filename', 'refcnt')])},
16660                {'call': 'swapon',
16661                 'reason': set([('filename', 'name'),
16662                                ('filename', 'refcnt')])},
16663                {'call': 'mq_open',
16664                 'reason': set([('filename', 'name'),
16665                                ('filename', 'refcnt')])},
16666                {'call': 'unlinkat',
16667                 'reason': set([('filename', 'name'),
16668                                ('filename', 'refcnt')])}],
16669  'sync_file_range': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
16670                      {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
16671                      {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
16672                      {'call': 'fremovexattr',
16673                       'reason': set([('fd', 'flags')])},
16674                      {'call': 'readahead', 'reason': set([('fd', 'flags')])},
16675                      {'call': 'getdents', 'reason': set([('fd', 'flags')])},
16676                      {'call': 'writev', 'reason': set([('fd', 'flags')])},
16677                      {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
16678                      {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
16679                      {'call': 'pread64', 'reason': set([('fd', 'flags')])},
16680                      {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
16681                      {'call': 'read', 'reason': set([('fd', 'flags')])},
16682                      {'call': 'fchown', 'reason': set([('fd', 'flags')])},
16683                      {'call': 'mq_timedreceive',
16684                       'reason': set([('fd', 'flags')])},
16685                      {'call': 'utime', 'reason': set([('fd', 'flags')])},
16686                      {'call': 'fsync', 'reason': set([('fd', 'flags')])},
16687                      {'call': 'bpf', 'reason': set([('fd', 'flags')])},
16688                      {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
16689                      {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
16690                      {'call': 'sendto', 'reason': set([('fd', 'flags')])},
16691                      {'call': 'tee', 'reason': set([('fd', 'flags')])},
```



```
16692                      {'call': 'lseek', 'reason': set([('fd', 'flags')])},
16693                      {'call': 'connect', 'reason': set([('fd', 'flags')])},
16694                      {'call': 'getsockname',
16695                       'reason': set([('fd', 'flags')])},
16696                      {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
16697                      {'call': 'flock', 'reason': set([('fd', 'flags')])},
16698                      {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
16699                      {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
16700                      {'call': 'accept4', 'reason': set([('fd', 'flags')])},
16701                      {'call': 'old_readdir',
16702                       'reason': set([('fd', 'flags')])},
16703                      {'call': 'inotify_rm_watch',
16704                       'reason': set([('fd', 'flags')])},
16705                      {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
16706                      {'call': 'inotify_add_watch',
16707                       'reason': set([('fd', 'flags')])},
16708                      {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
16709                      {'call': 'splice', 'reason': set([('fd', 'flags')])},
16710                      {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
16711                      {'call': 'preadv', 'reason': set([('fd', 'flags')])},
16712                      {'call': 'getpeername',
16713                       'reason': set([('fd', 'flags')])},
16714                      {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
16715                      {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
16716                      {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
16717                      {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
16718                      {'call': 'perf_event_open',
16719                       'reason': set([('fd', 'flags')])},
16720                      {'call': 'pwritev64v2',
16721                       'reason': set([('fd', 'flags')])},
16722                      {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
16723                      {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
16724                      {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
16725                      {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
16726                      {'call': 'mq_getsetattr',
16727                       'reason': set([('fd', 'flags')])},
16728                      {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
16729                      {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
16730                      {'call': 'listen', 'reason': set([('fd', 'flags')])},
16731                      {'call': 'copy_file_range',
16732                       'reason': set([('fd', 'flags')])},
16733                      {'call': 'mq_timedsend',
16734                       'reason': set([('fd', 'flags')])},
16735                      {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
16736                      {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
16737                      {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
16738                      {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
16739                      {'call': 'llseek', 'reason': set([('fd', 'flags')])},
16740                      {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
16741                      {'call': 'readv', 'reason': set([('fd', 'flags')])},
16742                      {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
16743                      {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
16744                      {'call': 'write', 'reason': set([('fd', 'flags')])},
16745                      {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
16746                      {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
16747                      {'call': 'bind', 'reason': set([('fd', 'flags')])},
16748                      {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
16749                      {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
16750  'syncfs': [{'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
16751             {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
16752             {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
16753             {'call': 'readahead', 'reason': set([('fd', 'flags')])},
16754             {'call': 'getdents', 'reason': set([('fd', 'flags')])},
16755             {'call': 'writev', 'reason': set([('fd', 'flags')])},
16756             {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
16757             {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
```

```
16758             {'call': 'pread64', 'reason': set([('fd', 'flags')])},
16759             {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
16760             {'call': 'read', 'reason': set([('fd', 'flags')])},
16761             {'call': 'fchown', 'reason': set([('fd', 'flags')])},
16762             {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
16763             {'call': 'utime', 'reason': set([('fd', 'flags')])},
16764             {'call': 'fsync', 'reason': set([('fd', 'flags')])},
16765             {'call': 'bpf', 'reason': set([('fd', 'flags')])},
16766             {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
16767             {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
16768             {'call': 'sendto', 'reason': set([('fd', 'flags')])},
16769             {'call': 'tee', 'reason': set([('fd', 'flags')])},
16770             {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
16771             {'call': 'lseek', 'reason': set([('fd', 'flags')])},
16772             {'call': 'connect', 'reason': set([('fd', 'flags')])},
16773             {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
16774             {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
16775             {'call': 'flock', 'reason': set([('fd', 'flags')])},
16776             {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
16777             {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
16778             {'call': 'accept4', 'reason': set([('fd', 'flags')])},
16779             {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
16780             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
16781             {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
16782             {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
16783             {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
16784             {'call': 'splice', 'reason': set([('fd', 'flags')])},
16785             {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
16786             {'call': 'preadv', 'reason': set([('fd', 'flags')])},
16787             {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
16788             {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
16789             {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
16790             {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
16791             {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
16792             {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
16793             {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
16794             {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
16795             {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
16796             {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
16797             {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
16798             {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
16799             {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
16800             {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
16801             {'call': 'listen', 'reason': set([('fd', 'flags')])},
16802             {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
16803             {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
16804             {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
16805             {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
16806             {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
16807             {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
16808             {'call': 'llseek', 'reason': set([('fd', 'flags')])},
16809             {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
16810             {'call': 'readv', 'reason': set([('fd', 'flags')])},
16811             {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
16812             {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
16813             {'call': 'write', 'reason': set([('fd', 'flags')])},
16814             {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
16815             {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
16816             {'call': 'bind', 'reason': set([('fd', 'flags')])},
16817             {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
16818             {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
16819  'sysfs': [{'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
16820            {'call': 'swapoff', 'reason': set([('filename', 'name')])},
16821            {'call': 'openat', 'reason': set([('filename', 'name')])},
16822            {'call': 'uselib', 'reason': set([('filename', 'name')])},
16823            {'call': 'renameat2', 'reason': set([('filename', 'name')])},
```



```
16824            {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
16825            {'call': 'quotactl', 'reason': set([('filename', 'name')])},
16826            {'call': 'acct', 'reason': set([('filename', 'name')])},
16827            {'call': 'open', 'reason': set([('filename', 'name')])},
16828            {'call': 'unlink', 'reason': set([('filename', 'name')])},
16829            {'call': 'rmdir', 'reason': set([('filename', 'name')])},
16830            {'call': 'swapon', 'reason': set([('filename', 'name')])},
16831            {'call': 'mq_open', 'reason': set([('filename', 'name')])},
16832            {'call': 'unlinkat', 'reason': set([('filename', 'name')])}],
16833  'sysinfo': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
16834              {'call': 'rt_sigtimedwait',
16835               'reason': set([('mm_segment_t', 'seg'),
16836                              ('timespec', 'tv_nsec')])},
16837              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
16838              {'call': 'mq_unlink', 'reason': set([('timespec', 'tv_nsec')])},
16839              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
16840              {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
16841              {'call': 'sched_getaffinity',
16842               'reason': set([('mm_segment_t', 'seg')])},
16843              {'call': 'sched_setparam',
16844               'reason': set([('mm_segment_t', 'seg')])},
16845              {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
16846              {'call': 'memfd_create',
16847               'reason': set([('timespec', 'tv_nsec')])},
16848              {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
16849              {'call': 'readlinkat', 'reason': set([('timespec', 'tv_nsec')])},
16850              {'call': 'io_getevents',
16851               'reason': set([('timespec', 'tv_nsec')])},
16852              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
16853              {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
16854              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
16855              {'call': 'mq_timedreceive',
16856               'reason': set([('mm_segment_t', 'seg'),
16857                              ('timespec', 'tv_nsec')])},
16858              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
16859              {'call': 'utime', 'reason': set([('timespec', 'tv_nsec')])},
16860              {'call': 'sched_setaffinity',
16861               'reason': set([('mm_segment_t', 'seg')])},
16862              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
16863              {'call': 'semtimedop',
16864               'reason': set([('mm_segment_t', 'seg'),
16865                              ('timespec', 'tv_nsec')])},
16866              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
16867              {'call': 'settimeofday',
16868               'reason': set([('timespec', 'tv_nsec')])},
16869              {'call': 'sched_rr_get_interval',
16870               'reason': set([('mm_segment_t', 'seg'),
16871                              ('timespec', 'tv_nsec')])},
16872              {'call': 'timerfd_gettime',
16873               'reason': set([('timespec', 'tv_nsec')])},
16874              {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
16875              {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
16876              {'call': 'rt_sigprocmask',
16877               'reason': set([('mm_segment_t', 'seg')])},
16878              {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
16879              {'call': 'sigaltstack',
16880               'reason': set([('mm_segment_t', 'seg')])},
16881              {'call': 'sched_setattr',
16882               'reason': set([('mm_segment_t', 'seg')])},
16883              {'call': 'migrate_pages',
16884               'reason': set([('mm_segment_t', 'seg')])},
16885              {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
16886              {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
16887              {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
16888              {'call': 'inotify_add_watch',
16889               'reason': set([('timespec', 'tv_nsec')])},
```

```
16890             {'call': 'timer_settime',
16891              'reason': set([('timespec', 'tv_nsec')])},
16892             {'call': 'ftruncate', 'reason': set([('timespec', 'tv_nsec')])},
16893             {'call': 'timer_gettime',
16894              'reason': set([('timespec', 'tv_nsec')])},
16895             {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
16896             {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
16897             {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
16898             {'call': 'perf_event_open',
16899              'reason': set([('mm_segment_t', 'seg')])},
16900             {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
16901             {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
16902             {'call': 'rt_sigaction',
16903              'reason': set([('mm_segment_t', 'seg')])},
16904             {'call': 'futimesat', 'reason': set([('timespec', 'tv_nsec')])},
16905             {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
16906             {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
16907             {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
16908             {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
16909             {'call': 'getpriority',
16910              'reason': set([('mm_segment_t', 'seg')])},
16911             {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
16912             {'call': 'nanosleep', 'reason': set([('timespec', 'tv_nsec')])},
16913             {'call': 'mq_getsetattr',
16914              'reason': set([('timespec', 'tv_nsec')])},
16915             {'call': 'faccessat', 'reason': set([('timespec', 'tv_nsec')])},
16916             {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
16917             {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
16918             {'call': 'get_robust_list',
16919              'reason': set([('mm_segment_t', 'seg')])},
16920             {'call': 'mq_timedsend',
16921              'reason': set([('mm_segment_t', 'seg'),
16922                             ('timespec', 'tv_nsec')])},
16923             {'call': 'sched_getscheduler',
16924              'reason': set([('mm_segment_t', 'seg')])},
16925             {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
16926             {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
16927             {'call': 'epoll_wait', 'reason': set([('timespec', 'tv_nsec')])},
16928             {'call': 'sched_getattr',
16929              'reason': set([('mm_segment_t', 'seg')])},
16930             {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
16931             {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
16932             {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
16933             {'call': 'timerfd_settime',
16934              'reason': set([('timespec', 'tv_nsec')])},
16935             {'call': 'sched_setscheduler',
16936              'reason': set([('mm_segment_t', 'seg')])},
16937             {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
16938             {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
16939             {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
16940             {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
16941             {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
16942             {'call': 'setpriority',
16943              'reason': set([('mm_segment_t', 'seg')])},
16944             {'call': 'mq_notify', 'reason': set([('timespec', 'tv_nsec')])},
16945             {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
16946             {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
16947             {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
16948             {'call': 'clock_nanosleep',
16949              'reason': set([('timespec', 'tv_nsec')])},
16950             {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
16951             {'call': 'sched_getparam',
16952              'reason': set([('mm_segment_t', 'seg')])},
16953             {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
16954             {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
16955             {'call': 'sendfile64', 'reason': set([('timespec', 'tv_nsec')])},
```



```
16956              {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
16957  'syslog': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
16958             {'call': 'rt_sigtimedwait',
16959              'reason': set([('mm_segment_t', 'seg')])},
16960             {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
16961             {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
16962             {'call': 'sched_getaffinity',
16963              'reason': set([('mm_segment_t', 'seg')])},
16964             {'call': 'sched_setparam',
16965              'reason': set([('mm_segment_t', 'seg')])},
16966             {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
16967             {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
16968             {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
16969             {'call': 'mq_timedreceive',
16970              'reason': set([('mm_segment_t', 'seg')])},
16971             {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
16972             {'call': 'sched_setaffinity',
16973              'reason': set([('mm_segment_t', 'seg')])},
16974             {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
16975             {'call': 'semtimedop', 'reason': set([('mm_segment_t', 'seg')])},
16976             {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
16977             {'call': 'sched_rr_get_interval',
16978              'reason': set([('mm_segment_t', 'seg')])},
16979             {'call': 'rt_sigprocmask',
16980              'reason': set([('mm_segment_t', 'seg')])},
16981             {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
16982             {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
16983             {'call': 'sched_setattr',
16984              'reason': set([('mm_segment_t', 'seg')])},
16985             {'call': 'migrate_pages',
16986              'reason': set([('mm_segment_t', 'seg')])},
16987             {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
16988             {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
16989             {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
16990             {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
16991             {'call': 'perf_event_open',
16992              'reason': set([('mm_segment_t', 'seg')])},
16993             {'call': 'rt_sigaction',
16994              'reason': set([('mm_segment_t', 'seg')])},
16995             {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
16996             {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
16997             {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
16998             {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
16999             {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
17000             {'call': 'get_robust_list',
17001              'reason': set([('mm_segment_t', 'seg')])},
17002             {'call': 'mq_timedsend',
17003              'reason': set([('mm_segment_t', 'seg')])},
17004             {'call': 'sched_getscheduler',
17005              'reason': set([('mm_segment_t', 'seg')])},
17006             {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
17007             {'call': 'sched_getattr',
17008              'reason': set([('mm_segment_t', 'seg')])},
17009             {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
17010             {'call': 'sched_setscheduler',
17011              'reason': set([('mm_segment_t', 'seg')])},
17012             {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
17013             {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
17014             {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
17015             {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
17016             {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
17017             {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
17018             {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
17019             {'call': 'sched_getparam',
17020              'reason': set([('mm_segment_t', 'seg')])}],
17021  'tee': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
```

```
17022          {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
17023          {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
17024          {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
17025          {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
17026          {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
17027          {'call': 'readahead', 'reason': set([('fd', 'flags')])},
17028          {'call': 'getdents', 'reason': set([('fd', 'flags')])},
17029          {'call': 'writev', 'reason': set([('fd', 'flags')])},
17030          {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
17031          {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
17032          {'call': 'pread64', 'reason': set([('fd', 'flags')])},
17033          {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
17034          {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
17035          {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
17036          {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
17037          {'call': 'read', 'reason': set([('fd', 'flags')])},
17038          {'call': 'fchown', 'reason': set([('fd', 'flags')])},
17039          {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
17040          {'call': 'utime', 'reason': set([('fd', 'flags')])},
17041          {'call': 'fsync', 'reason': set([('fd', 'flags')])},
17042          {'call': 'bpf', 'reason': set([('fd', 'flags')])},
17043          {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
17044          {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
17045          {'call': 'sendto', 'reason': set([('fd', 'flags')])},
17046          {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
17047          {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
17048          {'call': 'lseek', 'reason': set([('fd', 'flags')])},
17049          {'call': 'connect', 'reason': set([('fd', 'flags')])},
17050          {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
17051          {'call': 'epoll_ctl',
17052           'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
17053          {'call': 'flock',
17054           'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
17055          {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
17056          {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
17057          {'call': 'openat', 'reason': set([('file', 'f_mode')])},
17058          {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
17059          {'call': 'accept4',
17060           'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
17061          {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
17062          {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
17063          {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
17064          {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
17065          {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
17066          {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
17067          {'call': 'splice', 'reason': set([('fd', 'flags')])},
17068          {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
17069          {'call': 'preadv', 'reason': set([('fd', 'flags')])},
17070          {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
17071          {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
17072          {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
17073          {'call': 'socket', 'reason': set([('file', 'f_mode')])},
17074          {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
17075          {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
17076          {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
17077          {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
17078          {'call': 'perf_event_open',
17079           'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
17080          {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
17081          {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
17082          {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
17083          {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
17084          {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
17085          {'call': 'acct', 'reason': set([('file', 'f_mode')])},
17086          {'call': 'open', 'reason': set([('file', 'f_mode')])},
17087          {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
```



```
17088          {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
17089          {'call': 'dup', 'reason': set([('file', 'f_mode')])},
17090          {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
17091          {'call': 'setns', 'reason': set([('file', 'f_mode')])},
17092          {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
17093          {'call': 'listen', 'reason': set([('fd', 'flags')])},
17094          {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
17095          {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
17096          {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
17097          {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
17098          {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
17099          {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
17100          {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
17101          {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
17102          {'call': 'llseek', 'reason': set([('fd', 'flags')])},
17103          {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
17104          {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
17105          {'call': 'readv', 'reason': set([('fd', 'flags')])},
17106          {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
17107          {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
17108          {'call': 'write', 'reason': set([('fd', 'flags')])},
17109          {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
17110          {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
17111          {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
17112          {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
17113          {'call': 'bind', 'reason': set([('fd', 'flags')])},
17114          {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
17115          {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
17116   'tgkill': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
17117              {'call': 'rt_sigtimedwait',
17118               'reason': set([('task_struct', 'cred')])},
17119              {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
17120              {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
17121              {'call': 'sched_getaffinity',
17122               'reason': set([('task_struct', 'cred')])},
17123              {'call': 'sched_setparam',
17124               'reason': set([('task_struct', 'cred')])},
17125              {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
17126              {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
17127              {'call': 'mq_timedreceive',
17128               'reason': set([('task_struct', 'cred')])},
17129              {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
17130              {'call': 'sched_setaffinity',
17131               'reason': set([('task_struct', 'cred')])},
17132              {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
17133              {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
17134              {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
17135              {'call': 'sched_rr_get_interval',
17136               'reason': set([('task_struct', 'cred')])},
17137              {'call': 'rt_sigprocmask',
17138               'reason': set([('task_struct', 'cred')])},
17139              {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
17140              {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
17141              {'call': 'sched_setattr',
17142               'reason': set([('task_struct', 'cred')])},
17143              {'call': 'migrate_pages',
17144               'reason': set([('task_struct', 'cred')])},
17145              {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
17146              {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
17147              {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
17148              {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
17149              {'call': 'perf_event_open',
17150               'reason': set([('task_struct', 'cred')])},
17151              {'call': 'rt_sigaction',
17152               'reason': set([('task_struct', 'cred')])},
17153              {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
```

```
17154             {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
17155             {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
17156             {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
17157             {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
17158             {'call': 'get_robust_list',
17159              'reason': set([('task_struct', 'cred')])},
17160             {'call': 'mq_timedsend',
17161              'reason': set([('task_struct', 'cred')])},
17162             {'call': 'sched_getscheduler',
17163              'reason': set([('task_struct', 'cred')])},
17164             {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
17165             {'call': 'sched_getattr',
17166              'reason': set([('task_struct', 'cred')])},
17167             {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
17168             {'call': 'sched_setscheduler',
17169              'reason': set([('task_struct', 'cred')])},
17170             {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
17171             {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
17172             {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
17173             {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
17174             {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
17175             {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
17176             {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
17177             {'call': 'sched_getparam',
17178              'reason': set([('task_struct', 'cred')])}],
17179  'timer_create': [{'call': 'clock_getres',
17180                    'reason': set([('k_clock', 'timer_create')])},
17181                   {'call': 'timer_delete',
17182                    'reason': set([('k_clock', 'timer_create'),
17183                                   ('k_itimer', 'it_pid')])},
17184                   {'call': 'clock_gettime',
17185                    'reason': set([('k_clock', 'timer_create')])},
17186                   {'call': 'timer_settime',
17187                    'reason': set([('k_clock', 'timer_create'),
17188                                   ('k_itimer', 'it_pid')])},
17189                   {'call': 'timer_gettime',
17190                    'reason': set([('k_clock', 'timer_create'),
17191                                   ('k_itimer', 'it_pid')])},
17192                   {'call': 'clock_settime',
17193                    'reason': set([('k_clock', 'timer_create')])},
17194                   {'call': 'timer_getoverrun',
17195                    'reason': set([('k_itimer', 'it_pid')])},
17196                   {'call': 'clock_nanosleep',
17197                    'reason': set([('k_clock', 'timer_create')])},
17198                   {'call': 'clock_adjtime',
17199                    'reason': set([('k_clock', 'timer_create')])}],
17200  'timer_delete': [{'call': 'keyctl',
17201                    'reason': set([('task_struct', 'signal')])},
17202                   {'call': 'rt_sigtimedwait',
17203                    'reason': set([('task_struct', 'signal')])},
17204                   {'call': 'msgrcv',
17205                    'reason': set([('task_struct', 'signal')])},
17206                   {'call': 'kill',
17207                    'reason': set([('task_struct', 'signal')])},
17208                   {'call': 'clock_getres',
17209                    'reason': set([('k_clock', 'timer_del')])},
17210                   {'call': 'sched_getaffinity',
17211                    'reason': set([('task_struct', 'signal')])},
17212                   {'call': 'sched_setparam',
17213                    'reason': set([('task_struct', 'signal')])},
17214                   {'call': 'ioprio_set',
17215                    'reason': set([('task_struct', 'signal')])},
17216                   {'call': 'getppid',
17217                    'reason': set([('task_struct', 'signal')])},
17218                   {'call': 'mq_timedreceive',
17219                    'reason': set([('task_struct', 'signal')])},
```



```
17220                   {'call': 'capget',
17221                    'reason': set([('task_struct', 'signal')])},
17222                   {'call': 'sched_setaffinity',
17223                    'reason': set([('task_struct', 'signal')])},
17224                   {'call': 'signal',
17225                    'reason': set([('task_struct', 'signal')])},
17226                   {'call': 'semtimedop',
17227                    'reason': set([('task_struct', 'signal')])},
17228                   {'call': 'umount',
17229                    'reason': set([('task_struct', 'signal')])},
17230                   {'call': 'timer_create',
17231                    'reason': set([('k_clock', 'timer_del'),
17232                                   ('k_itimer', 'it_pid'),
17233                                   ('k_itimer', 'it_signal'),
17234                                   ('k_itimer', 'sigq')])},
17235                   {'call': 'clock_gettime',
17236                    'reason': set([('k_clock', 'timer_del')])},
17237                   {'call': 'sched_rr_get_interval',
17238                    'reason': set([('task_struct', 'signal')])},
17239                   {'call': 'rt_sigprocmask',
17240                    'reason': set([('task_struct', 'signal')])},
17241                   {'call': 'setsid',
17242                    'reason': set([('task_struct', 'signal')])},
17243                   {'call': 'sigaltstack',
17244                    'reason': set([('task_struct', 'signal')])},
17245                   {'call': 'sched_setattr',
17246                    'reason': set([('task_struct', 'signal')])},
17247                   {'call': 'migrate_pages',
17248                    'reason': set([('task_struct', 'signal')])},
17249                   {'call': 'getitimer',
17250                    'reason': set([('task_struct', 'signal')])},
17251                   {'call': 'setpgid',
17252                    'reason': set([('task_struct', 'signal')])},
17253                   {'call': 'timer_settime',
17254                    'reason': set([('k_clock', 'timer_del'),
17255                                   ('k_itimer', 'it_pid'),
17256                                   ('k_itimer', 'it_signal'),
17257                                   ('k_itimer', 'sigq')])},
17258                   {'call': 'timer_gettime',
17259                    'reason': set([('k_clock', 'timer_del'),
17260                                   ('k_itimer', 'it_pid'),
17261                                   ('k_itimer', 'it_signal'),
17262                                   ('k_itimer', 'sigq')])},
17263                   {'call': 'getsid',
17264                    'reason': set([('task_struct', 'signal')])},
17265                   {'call': 'prlimit64',
17266                    'reason': set([('task_struct', 'signal')])},
17267                   {'call': 'perf_event_open',
17268                    'reason': set([('task_struct', 'signal')])},
17269                   {'call': 'rt_sigaction',
17270                    'reason': set([('task_struct', 'signal')])},
17271                   {'call': 'getpgid',
17272                    'reason': set([('task_struct', 'signal')])},
17273                   {'call': 'clock_settime',
17274                    'reason': set([('k_clock', 'timer_del')])},
17275                   {'call': 'getpriority',
17276                    'reason': set([('task_struct', 'signal')])},
17277                   {'call': 'sigaction',
17278                    'reason': set([('task_struct', 'signal')])},
17279                   {'call': 'setns',
17280                    'reason': set([('task_struct', 'signal')])},
17281                   {'call': 'fork',
17282                    'reason': set([('task_struct', 'signal')])},
17283                   {'call': 'get_robust_list',
17284                    'reason': set([('task_struct', 'signal')])},
17285                   {'call': 'mq_timedsend',
```

```
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'sched_getscheduler',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'ptrace',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'sched_getattr',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'getrusage',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'sched_setscheduler',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'setitimer',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'ioprio_get',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'vfork',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'prctl',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'move_pages',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'setpriority',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'timer_getoverrun',
                  'reason': set([('k_itimer', 'it_pid'),
                                 ('k_itimer', 'it_signal'),
                                 ('k_itimer', 'sigq')])},
                 {'call': 'clone',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'clock_nanosleep',
                  'reason': set([('k_clock', 'timer_del')])},
                 {'call': 'sched_getparam',
                  'reason': set([('task_struct', 'signal')])},
                 {'call': 'clock_adjtime',
                  'reason': set([('k_clock', 'timer_del')])}],
 'timer_getoverrun': [{'call': 'keyctl',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'rt_sigtimedwait',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'msgrcv',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'kill',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'timer_delete',
                       'reason': set([('k_itimer', 'it_signal')])},
                      {'call': 'sched_getaffinity',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_setparam',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'ioprio_set',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'getppid',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'mq_timedreceive',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'capget',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_setaffinity',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'signal',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'semtimedop',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'umount',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'timer_create',
```



```
                       'reason': set([('k_itimer', 'it_signal')])},
                      {'call': 'sched_rr_get_interval',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'rt_sigprocmask',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'setsid',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sigaltstack',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_setattr',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'migrate_pages',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'getitimer',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'setpgid',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'timer_settime',
                       'reason': set([('k_itimer', 'it_signal')])},
                      {'call': 'timer_gettime',
                       'reason': set([('k_itimer', 'it_signal')])},
                      {'call': 'getsid',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'prlimit64',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'perf_event_open',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'rt_sigaction',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'getpgid',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'getpriority',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sigaction',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'setns',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'fork',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'get_robust_list',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'mq_timedsend',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_getscheduler',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'ptrace',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_getattr',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'getrusage',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'sched_setscheduler',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'setitimer',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'ioprio_get',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'vfork',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'prctl',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'move_pages',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'setpriority',
                       'reason': set([('task_struct', 'signal')])},
                      {'call': 'clone',
```

```
                     'reason': set([('task_struct', 'signal')])},
                    {'call': 'sched_getparam',
                     'reason': set([('task_struct', 'signal')])}],
 'timer_gettime': [{'call': 'clock_getres',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'timer_delete',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'timer_create',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_gettime',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'timer_settime',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_settime',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_nanosleep',
                    'reason': set([('k_clock', 'timer_get')])},
                   {'call': 'clock_adjtime',
                    'reason': set([('k_clock', 'timer_get')])}],
 'timer_settime': [{'call': 'clock_getres',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'timer_delete',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'timer_create',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_gettime',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'timer_gettime',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_settime',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_nanosleep',
                    'reason': set([('k_clock', 'timer_set')])},
                   {'call': 'clock_adjtime',
                    'reason': set([('k_clock', 'timer_set')])}],
 'timerfd_create': [{'call': 'timerfd_gettime',
                     'reason': set([('timerfd_ctx', 'clockid')])},
                    {'call': 'timerfd_settime',
                     'reason': set([('timerfd_ctx', 'clockid')])}],
 'timerfd_gettime': [{'call': 'timerfd_settime',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])},
                     {'call': 'timerfd_create',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])}],
 'timerfd_settime': [{'call': 'timerfd_gettime',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])},
                     {'call': 'timerfd_create',
                      'reason': set([('timerfd_ctx', 'expired'),
                                     ('timerfd_ctx', 'tintv')])}],
 'tkill': [{'call': 'keyctl', 'reason': set([('task_struct', 'cred')])},
           {'call': 'rt_sigtimedwait',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'msgrcv', 'reason': set([('task_struct', 'cred')])},
           {'call': 'kill', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getaffinity',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setparam',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'ioprio_set', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getppid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'mq_timedreceive',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'capget', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setaffinity',
```



```
            'reason': set([('task_struct', 'cred')])},
           {'call': 'signal', 'reason': set([('task_struct', 'cred')])},
           {'call': 'semtimedop', 'reason': set([('task_struct', 'cred')])},
           {'call': 'umount', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_rr_get_interval',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'rt_sigprocmask',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'setsid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sigaltstack', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setattr',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'migrate_pages',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'getitimer', 'reason': set([('task_struct', 'cred')])},
           {'call': 'setpgid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getsid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'prlimit64', 'reason': set([('task_struct', 'cred')])},
           {'call': 'perf_event_open',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'rt_sigaction', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getpgid', 'reason': set([('task_struct', 'cred')])},
           {'call': 'getpriority', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sigaction', 'reason': set([('task_struct', 'cred')])},
           {'call': 'setns', 'reason': set([('task_struct', 'cred')])},
           {'call': 'fork', 'reason': set([('task_struct', 'cred')])},
           {'call': 'get_robust_list',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'mq_timedsend', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getscheduler',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'ptrace', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getattr',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'getrusage', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_setscheduler',
            'reason': set([('task_struct', 'cred')])},
           {'call': 'setitimer', 'reason': set([('task_struct', 'cred')])},
           {'call': 'ioprio_get', 'reason': set([('task_struct', 'cred')])},
           {'call': 'vfork', 'reason': set([('task_struct', 'cred')])},
           {'call': 'prctl', 'reason': set([('task_struct', 'cred')])},
           {'call': 'move_pages', 'reason': set([('task_struct', 'cred')])},
           {'call': 'setpriority', 'reason': set([('task_struct', 'cred')])},
           {'call': 'clone', 'reason': set([('task_struct', 'cred')])},
           {'call': 'sched_getparam',
            'reason': set([('task_struct', 'cred')])}],
 'umount': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
            {'call': 'keyctl', 'reason': set([('task_struct', 'flags')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'msgrcv', 'reason': set([('task_struct', 'flags')])},
            {'call': 'eventfd2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'mq_unlink',
             'reason': set([('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'kill', 'reason': set([('task_struct', 'flags')])},
            {'call': 'swapoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'pivot_root',
             'reason': set([('mount', 'mnt_ns'),
                            ('path', 'dentry'),
```

```
                              ('path', 'mnt'),
                              ('vfsmount', 'mnt_flags'),
                              ('vfsmount', 'mnt_root')])},
            {'call': 'memfd_create',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'ioprio_set', 'reason': set([('task_struct', 'flags')])},
            {'call': 'remap_file_pages',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'dup3',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'getppid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'mq_timedreceive',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'capget', 'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setaffinity',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
            {'call': 'unshare',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'signal', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'semtimedop', 'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'epoll_create1',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'epoll_ctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'flock',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'openat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'lookup_dcookie',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'uselib',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'accept4',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'setsid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'sigaltstack',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setattr',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'socketpair',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'migrate_pages',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'getitimer', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setpgid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'getcwd',
             'reason': set([('mount', 'mnt_ns'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'getsid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'shmat',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'socket',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'pipe2',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'prlimit64', 'reason': set([('task_struct', 'flags')])},
            {'call': 'perf_event_open',
```



```
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('task_struct', 'flags')])},
            {'call': 'shmdt',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'quotactl',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('super_block', 's_flags')])},
            {'call': 'rt_sigaction',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'getpgid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'acct',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'open',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'getpriority',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sigaction', 'reason': set([('task_struct', 'flags')])},
            {'call': 'dup',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'setns',
             'reason': set([('nsproxy', 'mnt_ns'),
                            ('path', 'dentry'),
                            ('path', 'mnt'),
                            ('task_struct', 'flags')])},
            {'call': 'fork', 'reason': set([('task_struct', 'flags')])},
            {'call': 'get_robust_list',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'mq_timedsend',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_getscheduler',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'ptrace', 'reason': set([('task_struct', 'flags')])},
            {'call': 'shmctl',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'swapon',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('super_block', 's_flags')])},
            {'call': 'sched_getattr',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'getrusage', 'reason': set([('task_struct', 'flags')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setitimer', 'reason': set([('task_struct', 'flags')])},
            {'call': 'ioprio_get', 'reason': set([('task_struct', 'flags')])},
            {'call': 'vfork', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
            {'call': 'mmap_pgoff',
             'reason': set([('path', 'dentry'), ('path', 'mnt')])},
            {'call': 'prctl', 'reason': set([('task_struct', 'flags')])},
            {'call': 'move_pages', 'reason': set([('task_struct', 'flags')])},
            {'call': 'setpriority',
             'reason': set([('task_struct', 'flags')])},
            {'call': 'clone', 'reason': set([('task_struct', 'flags')])},
            {'call': 'mq_open',
             'reason': set([('path', 'dentry'),
                            ('path', 'mnt'),
                            ('vfsmount', 'mnt_flags'),
                            ('vfsmount', 'mnt_root')])},
            {'call': 'sched_getparam',
```

```
17682              'reason': set([('task_struct', 'flags')])},
17683             {'call': 'open_by_handle_at',
17684              'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
17685  'uname': [{'call': 'keyctl',
17686             'reason': set([('task_struct', 'personality')])},
17687            {'call': 'rt_sigtimedwait',
17688             'reason': set([('task_struct', 'personality')])},
17689            {'call': 'msgrcv',
17690             'reason': set([('task_struct', 'personality')])},
17691            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
17692            {'call': 'sched_getaffinity',
17693             'reason': set([('task_struct', 'personality')])},
17694            {'call': 'sched_setparam',
17695             'reason': set([('task_struct', 'personality')])},
17696            {'call': 'ioprio_set',
17697             'reason': set([('task_struct', 'personality')])},
17698            {'call': 'personality',
17699             'reason': set([('task_struct', 'personality')])},
17700            {'call': 'getppid',
17701             'reason': set([('task_struct', 'personality')])},
17702            {'call': 'mq_timedreceive',
17703             'reason': set([('task_struct', 'personality')])},
17704            {'call': 'capget',
17705             'reason': set([('task_struct', 'personality')])},
17706            {'call': 'sched_setaffinity',
17707             'reason': set([('task_struct', 'personality')])},
17708            {'call': 'signal',
17709             'reason': set([('task_struct', 'personality')])},
17710            {'call': 'semtimedop',
17711             'reason': set([('task_struct', 'personality')])},
17712            {'call': 'umount',
17713             'reason': set([('task_struct', 'personality')])},
17714            {'call': 'sched_rr_get_interval',
17715             'reason': set([('task_struct', 'personality')])},
17716            {'call': 'rt_sigprocmask',
17717             'reason': set([('task_struct', 'personality')])},
17718            {'call': 'setsid',
17719             'reason': set([('task_struct', 'personality')])},
17720            {'call': 'sigaltstack',
17721             'reason': set([('task_struct', 'personality')])},
17722            {'call': 'sched_setattr',
17723             'reason': set([('task_struct', 'personality')])},
17724            {'call': 'migrate_pages',
17725             'reason': set([('task_struct', 'personality')])},
17726            {'call': 'getitimer',
17727             'reason': set([('task_struct', 'personality')])},
17728            {'call': 'setpgid',
17729             'reason': set([('task_struct', 'personality')])},
17730            {'call': 'getsid',
17731             'reason': set([('task_struct', 'personality')])},
17732            {'call': 'prlimit64',
17733             'reason': set([('task_struct', 'personality')])},
17734            {'call': 'perf_event_open',
17735             'reason': set([('task_struct', 'personality')])},
17736            {'call': 'rt_sigaction',
17737             'reason': set([('task_struct', 'personality')])},
17738            {'call': 'getpgid',
17739             'reason': set([('task_struct', 'personality')])},
17740            {'call': 'getpriority',
17741             'reason': set([('task_struct', 'personality')])},
17742            {'call': 'sigaction',
17743             'reason': set([('task_struct', 'personality')])},
17744            {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
17745            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
17746            {'call': 'get_robust_list',
17747             'reason': set([('task_struct', 'personality')])},
```



```
17748            {'call': 'mq_timedsend',
17749             'reason': set([('task_struct', 'personality')])},
17750            {'call': 'sched_getscheduler',
17751             'reason': set([('task_struct', 'personality')])},
17752            {'call': 'ptrace',
17753             'reason': set([('task_struct', 'personality')])},
17754            {'call': 'sched_getattr',
17755             'reason': set([('task_struct', 'personality')])},
17756            {'call': 'getrusage',
17757             'reason': set([('task_struct', 'personality')])},
17758            {'call': 'sched_setscheduler',
17759             'reason': set([('task_struct', 'personality')])},
17760            {'call': 'setitimer',
17761             'reason': set([('task_struct', 'personality')])},
17762            {'call': 'ioprio_get',
17763             'reason': set([('task_struct', 'personality')])},
17764            {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
17765            {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
17766            {'call': 'move_pages',
17767             'reason': set([('task_struct', 'personality')])},
17768            {'call': 'setpriority',
17769             'reason': set([('task_struct', 'personality')])},
17770            {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
17771            {'call': 'sched_getparam',
17772             'reason': set([('task_struct', 'personality')])}],
17773  'unlink': [{'call': 'eventfd2',
17774              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17775             {'call': 'mq_unlink', 'reason': set([('dentry', 'd_inode')])},
17776             {'call': 'swapoff',
17777              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17778             {'call': 'pivot_root',
17779              'reason': set([('dentry', 'd_inode'),
17780                             ('path', 'dentry'),
17781                             ('path', 'mnt')])},
17782             {'call': 'memfd_create',
17783              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17784             {'call': 'remap_file_pages',
17785              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17786             {'call': 'dup3',
17787              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17788             {'call': 'unshare',
17789              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17790             {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
17791             {'call': 'epoll_create1',
17792              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17793             {'call': 'epoll_ctl',
17794              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17795             {'call': 'flock',
17796              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17797             {'call': 'openat',
17798              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17799             {'call': 'lookup_dcookie',
17800              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17801             {'call': 'uselib',
17802              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17803             {'call': 'renameat2', 'reason': set([('dentry', 'd_inode')])},
17804             {'call': 'accept4',
17805              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17806             {'call': 'socketpair',
17807              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17808             {'call': 'getcwd',
17809              'reason': set([('dentry', 'd_inode'),
17810                             ('path', 'dentry'),
17811                             ('path', 'mnt')])},
17812             {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
17813             {'call': 'shmat',
```

```
17814              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17815             {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
17816             {'call': 'socket',
17817              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17818             {'call': 'symlinkat', 'reason': set([('dentry', 'd_inode')])},
17819             {'call': 'pipe2',
17820              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17821             {'call': 'perf_event_open',
17822              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17823             {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
17824             {'call': 'shmdt',
17825              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17826             {'call': 'quotactl',
17827              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17828             {'call': 'acct',
17829              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17830             {'call': 'open',
17831              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17832             {'call': 'rmdir', 'reason': set([('dentry', 'd_inode')])},
17833             {'call': 'dup',
17834              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17835             {'call': 'setns',
17836              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17837             {'call': 'shmctl',
17838              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17839             {'call': 'swapon',
17840              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17841             {'call': 'mmap_pgoff',
17842              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17843             {'call': 'mq_open',
17844              'reason': set([('dentry', 'd_inode'),
17845                             ('path', 'dentry'),
17846                             ('path', 'mnt')])},
17847             {'call': 'unlinkat', 'reason': set([('dentry', 'd_inode')])},
17848             {'call': 'open_by_handle_at',
17849              'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
17850  'unlinkat': [{'call': 'eventfd2',
17851                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17852               {'call': 'mq_unlink', 'reason': set([('dentry', 'd_inode')])},
17853               {'call': 'swapoff',
17854                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17855               {'call': 'pivot_root',
17856                'reason': set([('dentry', 'd_inode'),
17857                               ('path', 'dentry'),
17858                               ('path', 'mnt')])},
17859               {'call': 'memfd_create',
17860                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17861               {'call': 'remap_file_pages',
17862                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17863               {'call': 'dup3',
17864                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17865               {'call': 'unshare',
17866                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17867               {'call': 'mkdirat', 'reason': set([('dentry', 'd_inode')])},
17868               {'call': 'epoll_create1',
17869                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17870               {'call': 'epoll_ctl',
17871                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17872               {'call': 'flock',
17873                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17874               {'call': 'openat',
17875                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17876               {'call': 'lookup_dcookie',
17877                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17878               {'call': 'uselib',
17879                'reason': set([('path', 'dentry'), ('path', 'mnt')])},
```



```
17880             {'call': 'renameat2', 'reason': set([('dentry', 'd_inode')])},
17881             {'call': 'accept4',
17882              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17883             {'call': 'socketpair',
17884              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17885             {'call': 'getcwd',
17886              'reason': set([('dentry', 'd_inode'),
17887                             ('path', 'dentry'),
17888                             ('path', 'mnt')])},
17889             {'call': 'ftruncate', 'reason': set([('dentry', 'd_inode')])},
17890             {'call': 'shmat',
17891              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17892             {'call': 'mknodat', 'reason': set([('dentry', 'd_inode')])},
17893             {'call': 'socket',
17894              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17895             {'call': 'symlinkat', 'reason': set([('dentry', 'd_inode')])},
17896             {'call': 'pipe2',
17897              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17898             {'call': 'perf_event_open',
17899              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17900             {'call': 'linkat', 'reason': set([('dentry', 'd_inode')])},
17901             {'call': 'shmdt',
17902              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17903             {'call': 'quotactl',
17904              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17905             {'call': 'acct',
17906              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17907             {'call': 'open',
17908              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17909             {'call': 'unlink', 'reason': set([('dentry', 'd_inode')])},
17910             {'call': 'rmdir', 'reason': set([('dentry', 'd_inode')])},
17911             {'call': 'dup',
17912              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17913             {'call': 'setns',
17914              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17915             {'call': 'shmctl',
17916              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17917             {'call': 'swapon',
17918              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17919             {'call': 'mmap_pgoff',
17920              'reason': set([('path', 'dentry'), ('path', 'mnt')])},
17921             {'call': 'mq_open',
17922              'reason': set([('dentry', 'd_inode'),
17923                             ('path', 'dentry'),
17924                             ('path', 'mnt')])},
17925             {'call': 'open_by_handle_at',
17926              'reason': set([('path', 'dentry'), ('path', 'mnt')])}],
17927  'uselib': [{'call': 'syncfs', 'reason': set([('super_block', 's_iflags')])},
17928             {'call': 'mq_unlink', 'reason': set([('vfsmount', 'mnt_flags')])},
17929             {'call': 'pivot_root',
17930              'reason': set([('vfsmount', 'mnt_flags')])},
17931             {'call': 'ustat', 'reason': set([('super_block', 's_iflags')])},
17932             {'call': 'umount',
17933              'reason': set([('super_block', 's_iflags'),
17934                             ('vfsmount', 'mnt_flags')])},
17935             {'call': 'getcwd', 'reason': set([('vfsmount', 'mnt_flags')])},
17936             {'call': 'quotactl',
17937              'reason': set([('super_block', 's_iflags')])},
17938             {'call': 'acct', 'reason': set([('vfsmount', 'mnt_flags')])},
17939             {'call': 'swapon', 'reason': set([('super_block', 's_iflags')])},
17940             {'call': 'mq_open', 'reason': set([('vfsmount', 'mnt_flags')])}],
17941  'ustat': [{'call': 'syncfs', 'reason': set([('super_block', 's_root')])},
17942            {'call': 'umount', 'reason': set([('super_block', 's_root')])},
17943            {'call': 'quotactl', 'reason': set([('super_block', 's_root')])},
17944            {'call': 'swapon', 'reason': set([('super_block', 's_root')])}],
17945  'utime': [{'call': 'rt_sigtimedwait',
```

```
17946             'reason': set([('timespec', 'tv_nsec')])},
17947            {'call': 'mq_unlink', 'reason': set([('timespec', 'tv_nsec')])},
17948            {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
17949            {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
17950            {'call': 'memfd_create', 'reason': set([('timespec', 'tv_nsec')])},
17951            {'call': 'readlinkat', 'reason': set([('timespec', 'tv_nsec')])},
17952            {'call': 'io_getevents', 'reason': set([('timespec', 'tv_nsec')])},
17953            {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
17954            {'call': 'mq_timedreceive',
17955             'reason': set([('timespec', 'tv_nsec')])},
17956            {'call': 'semtimedop', 'reason': set([('timespec', 'tv_nsec')])},
17957            {'call': 'settimeofday', 'reason': set([('timespec', 'tv_nsec')])},
17958            {'call': 'sched_rr_get_interval',
17959             'reason': set([('timespec', 'tv_nsec')])},
17960            {'call': 'timerfd_gettime',
17961             'reason': set([('timespec', 'tv_nsec')])},
17962            {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
17963            {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
17964            {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
17965            {'call': 'inotify_add_watch',
17966             'reason': set([('timespec', 'tv_nsec')])},
17967            {'call': 'timer_settime',
17968             'reason': set([('timespec', 'tv_nsec')])},
17969            {'call': 'ftruncate', 'reason': set([('timespec', 'tv_nsec')])},
17970            {'call': 'timer_gettime',
17971             'reason': set([('timespec', 'tv_nsec')])},
17972            {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
17973            {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
17974            {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
17975            {'call': 'futimesat', 'reason': set([('timespec', 'tv_nsec')])},
17976            {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
17977            {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
17978            {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
17979            {'call': 'nanosleep', 'reason': set([('timespec', 'tv_nsec')])},
17980            {'call': 'mq_getsetattr',
17981             'reason': set([('timespec', 'tv_nsec')])},
17982            {'call': 'faccessat', 'reason': set([('timespec', 'tv_nsec')])},
17983            {'call': 'mq_timedsend', 'reason': set([('timespec', 'tv_nsec')])},
17984            {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
17985            {'call': 'epoll_wait', 'reason': set([('timespec', 'tv_nsec')])},
17986            {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
17987            {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
17988            {'call': 'timerfd_settime',
17989             'reason': set([('timespec', 'tv_nsec')])},
17990            {'call': 'mq_notify', 'reason': set([('timespec', 'tv_nsec')])},
17991            {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
17992            {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
17993            {'call': 'clock_nanosleep',
17994             'reason': set([('timespec', 'tv_nsec')])},
17995            {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
17996            {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
17997            {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
17998            {'call': 'sendfile64', 'reason': set([('timespec', 'tv_nsec')])},
17999            {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
18000  'utimensat': [{'call': 'rt_sigtimedwait',
18001                 'reason': set([('timespec', 'tv_nsec')])},
18002                {'call': 'mq_unlink',
18003                 'reason': set([('timespec', 'tv_nsec')])},
18004                {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
18005                {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
18006                {'call': 'memfd_create',
18007                 'reason': set([('timespec', 'tv_nsec')])},
18008                {'call': 'readlinkat',
18009                 'reason': set([('timespec', 'tv_nsec')])},
18010                {'call': 'io_getevents',
18011                 'reason': set([('timespec', 'tv_nsec')])},
```



```
18012                {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
18013                {'call': 'mq_timedreceive',
18014                 'reason': set([('timespec', 'tv_nsec')])},
18015                {'call': 'utime', 'reason': set([('timespec', 'tv_nsec')])},
18016                {'call': 'semtimedop',
18017                 'reason': set([('timespec', 'tv_nsec')])},
18018                {'call': 'settimeofday',
18019                 'reason': set([('timespec', 'tv_nsec')])},
18020                {'call': 'sched_rr_get_interval',
18021                 'reason': set([('timespec', 'tv_nsec')])},
18022                {'call': 'timerfd_gettime',
18023                 'reason': set([('timespec', 'tv_nsec')])},
18024                {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
18025                {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
18026                {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
18027                {'call': 'inotify_add_watch',
18028                 'reason': set([('timespec', 'tv_nsec')])},
18029                {'call': 'timer_settime',
18030                 'reason': set([('timespec', 'tv_nsec')])},
18031                {'call': 'ftruncate',
18032                 'reason': set([('timespec', 'tv_nsec')])},
18033                {'call': 'timer_gettime',
18034                 'reason': set([('timespec', 'tv_nsec')])},
18035                {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
18036                {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
18037                {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
18038                {'call': 'futimesat',
18039                 'reason': set([('timespec', 'tv_nsec')])},
18040                {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
18041                {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
18042                {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
18043                {'call': 'nanosleep',
18044                 'reason': set([('timespec', 'tv_nsec')])},
18045                {'call': 'mq_getsetattr',
18046                 'reason': set([('timespec', 'tv_nsec')])},
18047                {'call': 'faccessat',
18048                 'reason': set([('timespec', 'tv_nsec')])},
18049                {'call': 'mq_timedsend',
18050                 'reason': set([('timespec', 'tv_nsec')])},
18051                {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
18052                {'call': 'epoll_wait',
18053                 'reason': set([('timespec', 'tv_nsec')])},
18054                {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
18055                {'call': 'fstat', 'reason': set([('timespec', 'tv_nsec')])},
18056                {'call': 'timerfd_settime',
18057                 'reason': set([('timespec', 'tv_nsec')])},
18058                {'call': 'mq_notify',
18059                 'reason': set([('timespec', 'tv_nsec')])},
18060                {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
18061                {'call': 'newfstat', 'reason': set([('timespec', 'tv_nsec')])},
18062                {'call': 'clock_nanosleep',
18063                 'reason': set([('timespec', 'tv_nsec')])},
18064                {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
18065                {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
18066                {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
18067                {'call': 'sendfile64',
18068                 'reason': set([('timespec', 'tv_nsec')])},
18069                {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
18070  'vmsplice': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
18071               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
18072               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
18073               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
18074               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
18075               {'call': 'readahead', 'reason': set([('fd', 'flags')])},
18076               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
18077               {'call': 'writev', 'reason': set([('fd', 'flags')])},
```

```
18078                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
18079                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
18080                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
18081                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
18082                {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
18083                {'call': 'remap_file_pages',
18084                 'reason': set([('file', 'f_mode')])},
18085                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
18086                {'call': 'read', 'reason': set([('fd', 'flags')])},
18087                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
18088                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
18089                {'call': 'utime', 'reason': set([('fd', 'flags')])},
18090                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
18091                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
18092                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
18093                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
18094                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
18095                {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
18096                {'call': 'tee', 'reason': set([('fd', 'flags')])},
18097                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
18098                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
18099                {'call': 'connect', 'reason': set([('fd', 'flags')])},
18100                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
18101                {'call': 'epoll_ctl',
18102                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
18103                {'call': 'flock',
18104                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
18105                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
18106                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
18107                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
18108                {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
18109                {'call': 'accept4',
18110                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
18111                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
18112                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
18113                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
18114                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
18115                {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
18116                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
18117                {'call': 'splice', 'reason': set([('fd', 'flags')])},
18118                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
18119                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
18120                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
18121                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
18122                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
18123                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
18124                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
18125                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
18126                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
18127                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
18128                {'call': 'perf_event_open',
18129                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
18130                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
18131                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
18132                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
18133                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
18134                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
18135                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
18136                {'call': 'open', 'reason': set([('file', 'f_mode')])},
18137                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
18138                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
18139                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
18140                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
18141                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
18142                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
18143                {'call': 'listen', 'reason': set([('fd', 'flags')])},
```



```
18144                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
18145                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
18146                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
18147                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
18148                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
18149                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
18150                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
18151                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
18152                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
18153                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
18154                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
18155                {'call': 'readv', 'reason': set([('fd', 'flags')])},
18156                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
18157                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
18158                {'call': 'write', 'reason': set([('fd', 'flags')])},
18159                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
18160                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
18161                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
18162                {'call': 'open_by_handle_at',
18163                 'reason': set([('file', 'f_mode')])},
18164                {'call': 'bind', 'reason': set([('fd', 'flags')])},
18165                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
18166                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
18167  'write': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
18168            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
18169            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
18170            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
18171            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
18172            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
18173            {'call': 'writev', 'reason': set([('fd', 'flags')])},
18174            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
18175            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
18176            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
18177            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
18178            {'call': 'read', 'reason': set([('fd', 'flags')])},
18179            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
18180            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
18181            {'call': 'utime', 'reason': set([('fd', 'flags')])},
18182            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
18183            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
18184            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
18185            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
18186            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
18187            {'call': 'tee', 'reason': set([('fd', 'flags')])},
18188            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
18189            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
18190            {'call': 'connect', 'reason': set([('fd', 'flags')])},
18191            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
18192            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
18193            {'call': 'flock', 'reason': set([('fd', 'flags')])},
18194            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
18195            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
18196            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
18197            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
18198            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
18199            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
18200            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
18201            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
18202            {'call': 'splice', 'reason': set([('fd', 'flags')])},
18203            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
18204            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
18205            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
18206            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
18207            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
18208            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
18209            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
```

```
18210             {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
18211             {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
18212             {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
18213             {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
18214             {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
18215             {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
18216             {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
18217             {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
18218             {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
18219             {'call': 'listen', 'reason': set([('fd', 'flags')])},
18220             {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
18221             {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
18222             {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
18223             {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
18224             {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
18225             {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
18226             {'call': 'llseek', 'reason': set([('fd', 'flags')])},
18227             {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
18228             {'call': 'readv', 'reason': set([('fd', 'flags')])},
18229             {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
18230             {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
18231             {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
18232             {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
18233             {'call': 'bind', 'reason': set([('fd', 'flags')])},
18234             {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
18235             {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}]}
```

```
**********************************************************
  194619 Fri Dec 21 15:00:34 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/without_struct
s/implicit_dependencies.pretty
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
{'acct': set(['accept4',
              'dup',
              'dup3',
              'epoll_create1',
              'epoll_ctl',
              'eventfd2',
              'fadvise64_64',
              'flock',
              'memfd_create',
              'mmap_pgoff',
              'mq_open',
              'msync',
              'open',
              'open_by_handle_at',
              'openat',
              'perf_event_open',
              'pipe2',
              'remap_file_pages',
              'setns',
              'shmat',
              'shmctl',
              'shmdt',
              'socket',
              'socketpair',
              'swapoff',
              'swapon',
              'uselib']),
 'alarm': set(['adjtimex',
               'clock_adjtime',
               'getitimer',
               'getrusage',
               'ppoll',
               'select',
               'setitimer',
               'settimeofday',
               'wait4',
               'waitid']),
 'bpf': set(['accept4',
             'acct',
             'capget',
             'clone',
             'dup',
             'dup3',
             'epoll_create1',
             'epoll_ctl',
             'eventfd2',
             'flock',
             'fork',
             'get_robust_list',
             'getitimer',
             'getpgid',
             'getppid',
             'getpriority',
             'getrusage',
             'getsid',
             'ioperm',
             'iopl',
             'ioprio_get',
             'ioprio_set',
```



```
             'keyctl',
             'kill',
             'memfd_create',
             'migrate_pages',
             'mmap_pgoff',
             'move_pages',
             'mq_open',
             'mq_timedreceive',
             'mq_timedsend',
             'msgrcv',
             'msync',
             'open',
             'open_by_handle_at',
             'openat',
             'perf_event_open',
             'pipe2',
             'prctl',
             'prlimit64',
             'ptrace',
             'remap_file_pages',
             'rt_sigaction',
             'rt_sigprocmask',
             'rt_sigtimedwait',
             'sched_getaffinity',
             'sched_getattr',
             'sched_getparam',
             'sched_getscheduler',
             'sched_rr_get_interval',
             'sched_setaffinity',
             'sched_setattr',
             'sched_setparam',
             'sched_setscheduler',
             'semtimedop',
             'setitimer',
             'setns',
             'setpgid',
             'setpriority',
             'setsid',
             'shmat',
             'shmctl',
             'shmdt',
             'sigaction',
             'sigaltstack',
             'signal',
             'socket',
             'socketpair',
             'swapoff',
             'swapon',
             'umount',
             'uselib',
             'vfork']),
 'brk': set(['get_mempolicy',
             'getrusage',
             'io_cancel',
             'io_destroy',
             'io_getevents',
             'io_setup',
             'madvise',
             'mbind',
             'migrate_pages',
             'mincore',
             'mlockall',
             'modify_ldt',
             'move_pages',
             'mprotect',
             'mremap',
```

```
126             'msync',
127             'munlock',
128             'munlockall',
129             'pkey_mprotect',
130             'prctl',
131             'remap_file_pages',
132             'shmdt',
133             'swapoff']),
134  'clock_adjtime': set(['clock_getres',
135                        'clock_gettime',
136                        'clock_nanosleep',
137                        'clock_settime',
138                        'timer_create',
139                        'timer_delete',
140                        'timer_gettime',
141                        'timer_settime']),
142  'clock_nanosleep': set(['clock_adjtime',
143                          'clock_getres',
144                          'clock_gettime',
145                          'clock_settime',
146                          'epoll_wait',
147                          'faccessat',
148                          'fadvise64_64',
149                          'fchmod',
150                          'fchmodat',
151                          'fchown',
152                          'fchownat',
153                          'ftruncate',
154                          'futex',
155                          'futimesat',
156                          'inotify_add_watch',
157                          'io_getevents',
158                          'ioctl',
159                          'linkat',
160                          'memfd_create',
161                          'mq_getsetattr',
162                          'mq_notify',
163                          'mq_timedreceive',
164                          'mq_timedsend',
165                          'mq_unlink',
166                          'nanosleep',
167                          'poll',
168                          'ppoll',
169                          'pselect6',
170                          'readlinkat',
171                          'recvmmsg',
172                          'rt_sigtimedwait',
173                          'sched_rr_get_interval',
174                          'select',
175                          'semtimedop',
176                          'sendfile',
177                          'sendfile64',
178                          'settimeofday',
179                          'stime',
180                          'swapoff',
181                          'swapon',
182                          'timer_create',
183                          'timer_delete',
184                          'timer_gettime',
185                          'timer_settime',
186                          'timerfd_gettime',
187                          'timerfd_settime',
188                          'unlink',
189                          'unlinkat',
190                          'uselib',
191                          'utime']),
```



```
192  'clock_settime': set(['clock_adjtime',
193                        'clock_getres',
194                        'clock_gettime',
195                        'clock_nanosleep',
196                        'timer_create',
197                        'timer_delete',
198                        'timer_gettime',
199                        'timer_settime']),
200  'copy_file_range': set(['accept4',
201                          'bind',
202                          'bpf',
203                          'connect',
204                          'epoll_ctl',
205                          'epoll_wait',
206                          'fadvise64_64',
207                          'fallocate',
208                          'fchdir',
209                          'fchmod',
210                          'fchown',
211                          'fcntl',
212                          'fcntl64',
213                          'fdatasync',
214                          'fgetxattr',
215                          'flistxattr',
216                          'flock',
217                          'fremovexattr',
218                          'fsetxattr',
219                          'fstatfs',
220                          'fstatfs64',
221                          'fsync',
222                          'ftruncate',
223                          'futimesat',
224                          'getdents',
225                          'getdents64',
226                          'getpeername',
227                          'getsockname',
228                          'getsockopt',
229                          'inotify_add_watch',
230                          'inotify_rm_watch',
231                          'ioctl',
232                          'listen',
233                          'llseek',
234                          'lseek',
235                          'mq_getsetattr',
236                          'mq_notify',
237                          'mq_timedreceive',
238                          'mq_timedsend',
239                          'old_readdir',
240                          'perf_event_open',
241                          'pread64',
242                          'preadv',
243                          'preadv2',
244                          'preadv64',
245                          'preadv64v2',
246                          'pwrite64',
247                          'pwritev',
248                          'pwritev2',
249                          'pwritev64',
250                          'pwritev64v2',
251                          'read',
252                          'readahead',
253                          'readv',
254                          'recvfrom',
255                          'sendfile',
256                          'sendfile64',
257                          'sendto',
```

```
                        'setsockopt',
                        'shutdown',
                        'signalfd4',
                        'splice',
                        'sync_file_range',
                        'syncfs',
                        'tee',
                        'utime',
                        'utimensat',
                        'vmsplice',
                        'write',
                        'writev']),
 'delete_module': set(['finit_module', 'init_module']),
 'dup3': set(['dup2', 'select', 'unshare']),
 'epoll_ctl': set(['accept4',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'epoll_create1',
                   'epoll_wait',
                   'fadvise64_64',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'fgetxattr',
                   'flistxattr',
                   'flock',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'getdents',
                   'getdents64',
                   'getpeername',
                   'getsockname',
                   'getsockopt',
                   'inotify_add_watch',
                   'inotify_rm_watch',
                   'ioctl',
                   'listen',
                   'llseek',
                   'lseek',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'old_readdir',
                   'perf_event_open',
                   'pread64',
                   'preadv',
                   'preadv2',
                   'preadv64',
                   'preadv64v2',
                   'pwrite64',
                   'pwritev',
                   'pwritev2',
                   'pwritev64',
                   'pwritev64v2',
```



```
                   'read',
                   'readahead',
                   'readv',
                   'recvfrom',
                   'sendfile',
                   'sendfile64',
                   'sendto',
                   'setsockopt',
                   'shutdown',
                   'signalfd4',
                   'splice',
                   'sync_file_range',
                   'syncfs',
                   'tee',
                   'utime',
                   'utimensat',
                   'vmsplice',
                   'write',
                   'writev']),
 'epoll_wait': set(['accept4',
                    'bind',
                    'bpf',
                    'capget',
                    'clone',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'fadvise64_64',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fork',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'get_robust_list',
                    'getdents',
                    'getdents64',
                    'getitimer',
                    'getpeername',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'ioperm',
                    'iopl',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
```

```
                    'kill',
                    'listen',
                    'llseek',
                    'lseek',
                    'migrate_pages',
                    'move_pages',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'old_readdir',
                    'perf_event_open',
                    'prctl',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'prlimit64',
                    'ptrace',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'setsockopt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'umount',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
```



```
                   'write',
                   'writev']),
 'fadvise64_64': set(['accept4',
                      'bind',
                      'bpf',
                      'connect',
                      'copy_file_range',
                      'epoll_ctl',
                      'epoll_wait',
                      'faccessat',
                      'fallocate',
                      'fchdir',
                      'fchmod',
                      'fchmodat',
                      'fchown',
                      'fchownat',
                      'fcntl',
                      'fcntl64',
                      'fdatasync',
                      'fgetxattr',
                      'flistxattr',
                      'flock',
                      'fremovexattr',
                      'fsetxattr',
                      'fstatfs',
                      'fstatfs64',
                      'fsync',
                      'ftruncate',
                      'futimesat',
                      'getdents',
                      'getdents64',
                      'getpeername',
                      'getsockname',
                      'getsockopt',
                      'inotify_add_watch',
                      'inotify_rm_watch',
                      'ioctl',
                      'linkat',
                      'listen',
                      'llseek',
                      'lseek',
                      'memfd_create',
                      'mq_getsetattr',
                      'mq_notify',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'mq_unlink',
                      'old_readdir',
                      'perf_event_open',
                      'pread64',
                      'preadv',
                      'preadv2',
                      'preadv64',
                      'preadv64v2',
                      'pwrite64',
                      'pwritev',
                      'pwritev2',
                      'pwritev64',
                      'pwritev64v2',
                      'read',
                      'readahead',
                      'readlinkat',
                      'readv',
                      'recvfrom',
                      'sendfile',
                      'sendfile64',
```

```
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'swapoff',
                    'swapon',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'unlink',
                    'unlinkat',
                    'uselib',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
'fallocate': set(['accept4',
                  'bind',
                  'bpf',
                  'connect',
                  'copy_file_range',
                  'epoll_ctl',
                  'epoll_wait',
                  'fadvise64_64',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
                  'fremovexattr',
                  'fsetxattr',
                  'fstatfs',
                  'fstatfs64',
                  'fsync',
                  'ftruncate',
                  'futimesat',
                  'getdents',
                  'getdents64',
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'inotify_rm_watch',
                  'ioctl',
                  'listen',
                  'llseek',
                  'lseek',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'old_readdir',
                  'perf_event_open',
                  'pread64',
                  'preadv',
                  'preadv2',
                  'preadv64',
                  'preadv64v2',
                  'pwrite64',
                  'pwritev',
```



```
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
'fchdir': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fadvise64_64',
               'fallocate',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
```

```
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'syncfs',
                'tee',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchmod': set(['accept4',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'epoll_ctl',
                'epoll_wait',
                'fadvise64_64',
                'fallocate',
                'fchdir',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
```



```
                'perf_event_open',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'syncfs',
                'tee',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fchown': set(['accept4',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'epoll_ctl',
                'epoll_wait',
                'fadvise64_64',
                'fallocate',
                'fchdir',
                'fchmod',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'mq_getsetattr',
```

```
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'perf_event_open',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'syncfs',
                'tee',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fcntl': set(['accept4',
               'acct',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'epoll_wait',
               'eventfd2',
               'fadvise64_64',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
```



```
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'ioctl',
               'listen',
               'llseek',
               'lseek',
               'memfd_create',
               'mmap_pgoff',
               'mq_getsetattr',
               'mq_notify',
               'mq_open',
               'mq_timedreceive',
               'mq_timedsend',
               'msync',
               'old_readdir',
               'open',
               'open_by_handle_at',
               'openat',
               'perf_event_open',
               'pipe2',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
               'readv',
               'recvfrom',
               'remap_file_pages',
               'sendfile',
               'sendfile64',
               'sendto',
               'setns',
               'setsockopt',
               'shmat',
               'shmctl',
               'shmdt',
               'shutdown',
               'signalfd4',
               'socket',
               'socketpair',
               'splice',
               'swapoff',
               'swapon',
               'sync_file_range',
               'syncfs',
               'tee',
               'uselib',
               'utime',
               'utimensat',
               'vmsplice',
               'write',
               'writev']),
 'fcntl64': set(['accept4',
                 'acct',
                 'bind',
                 'bpf',
                 'connect',
```

```
                'copy_file_range',
                'dup',
                'dup3',
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fadvise64_64',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'msync',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
```



```
                'sendto',
                'setns',
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'splice',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'fgetxattr': set(['accept4',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'epoll_ctl',
                   'epoll_wait',
                   'fadvise64_64',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
                   'fcntl',
                   'fcntl64',
                   'fdatasync',
                   'flistxattr',
                   'flock',
                   'fremovexattr',
                   'fsetxattr',
                   'fstatfs',
                   'fstatfs64',
                   'fsync',
                   'ftruncate',
                   'futimesat',
                   'getdents',
                   'getdents64',
                   'getpeername',
                   'getsockname',
                   'getsockopt',
                   'inotify_add_watch',
                   'inotify_rm_watch',
                   'ioctl',
                   'listen',
                   'llseek',
                   'lseek',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'old_readdir',
                   'perf_event_open',
                   'pread64',
                   'preadv',
                   'preadv2',
```

```
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'finit_module': set(['delete_module', 'init_module']),
 'flistxattr': set(['accept4',
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fadvise64_64',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
```



```
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'flock': set(['accept4',
               'acct',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'dup',
               'dup3',
               'epoll_create1',
               'epoll_ctl',
               'epoll_wait',
               'eventfd2',
               'fadvise64_64',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
```

```
1182                 'inotify_rm_watch',
1183                 'ioctl',
1184                 'listen',
1185                 'llseek',
1186                 'lseek',
1187                 'memfd_create',
1188                 'mmap_pgoff',
1189                 'mq_getsetattr',
1190                 'mq_notify',
1191                 'mq_open',
1192                 'mq_timedreceive',
1193                 'mq_timedsend',
1194                 'msync',
1195                 'old_readdir',
1196                 'open',
1197                 'open_by_handle_at',
1198                 'openat',
1199                 'perf_event_open',
1200                 'pipe2',
1201                 'pread64',
1202                 'preadv',
1203                 'preadv2',
1204                 'preadv64',
1205                 'preadv64v2',
1206                 'pwrite64',
1207                 'pwritev',
1208                 'pwritev2',
1209                 'pwritev64',
1210                 'pwritev64v2',
1211                 'quotactl',
1212                 'read',
1213                 'readahead',
1214                 'readv',
1215                 'recvfrom',
1216                 'remap_file_pages',
1217                 'sendfile',
1218                 'sendfile64',
1219                 'sendto',
1220                 'setns',
1221                 'setsockopt',
1222                 'shmat',
1223                 'shmctl',
1224                 'shmdt',
1225                 'shutdown',
1226                 'signalfd4',
1227                 'socket',
1228                 'socketpair',
1229                 'splice',
1230                 'swapoff',
1231                 'swapon',
1232                 'sync_file_range',
1233                 'syncfs',
1234                 'tee',
1235                 'umount',
1236                 'uselib',
1237                 'ustat',
1238                 'utime',
1239                 'utimensat',
1240                 'vmsplice',
1241                 'write',
1242                 'writev']),
1243  'fremovexattr': set(['accept4',
1244                       'bind',
1245                       'bpf',
1246                       'connect',
1247                       'copy_file_range',
```



```
1248                       'epoll_ctl',
1249                       'epoll_wait',
1250                       'fadvise64_64',
1251                       'fallocate',
1252                       'fchdir',
1253                       'fchmod',
1254                       'fchown',
1255                       'fcntl',
1256                       'fcntl64',
1257                       'fdatasync',
1258                       'fgetxattr',
1259                       'flistxattr',
1260                       'flock',
1261                       'fsetxattr',
1262                       'fstatfs',
1263                       'fstatfs64',
1264                       'fsync',
1265                       'ftruncate',
1266                       'futimesat',
1267                       'getdents',
1268                       'getdents64',
1269                       'getpeername',
1270                       'getsockname',
1271                       'getsockopt',
1272                       'inotify_add_watch',
1273                       'inotify_rm_watch',
1274                       'ioctl',
1275                       'listen',
1276                       'llseek',
1277                       'lseek',
1278                       'mq_getsetattr',
1279                       'mq_notify',
1280                       'mq_timedreceive',
1281                       'mq_timedsend',
1282                       'old_readdir',
1283                       'perf_event_open',
1284                       'pread64',
1285                       'preadv',
1286                       'preadv2',
1287                       'preadv64',
1288                       'preadv64v2',
1289                       'pwrite64',
1290                       'pwritev',
1291                       'pwritev2',
1292                       'pwritev64',
1293                       'pwritev64v2',
1294                       'read',
1295                       'readahead',
1296                       'readv',
1297                       'recvfrom',
1298                       'sendfile',
1299                       'sendfile64',
1300                       'sendto',
1301                       'setsockopt',
1302                       'shutdown',
1303                       'signalfd4',
1304                       'splice',
1305                       'sync_file_range',
1306                       'syncfs',
1307                       'tee',
1308                       'utime',
1309                       'utimensat',
1310                       'vmsplice',
1311                       'write',
1312                       'writev']),
1313  'fsetxattr': set(['accept4',
```

```
                    'bind',
                    'bpf',
                    'connect',
                    'copy_file_range',
                    'epoll_ctl',
                    'epoll_wait',
                    'fadvise64_64',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
```



```
                    'vmsplice',
                    'write',
                    'writev']),
    'fstat': set(['lstat', 'stat']),
    'fstatfs': set(['fstatfs64', 'statfs', 'statfs64', 'ustat']),
    'fstatfs64': set(['fstatfs', 'statfs', 'statfs64', 'ustat']),
    'ftruncate': set(['accept4',
                    'acct',
                    'dup',
                    'dup3',
                    'epoll_create1',
                    'epoll_ctl',
                    'eventfd2',
                    'faccessat',
                    'fadvise64_64',
                    'fchmod',
                    'fchmodat',
                    'fchown',
                    'fchownat',
                    'flock',
                    'inotify_add_watch',
                    'ioctl',
                    'linkat',
                    'memfd_create',
                    'mmap_pgoff',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_open',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'mq_unlink',
                    'msync',
                    'open',
                    'open_by_handle_at',
                    'openat',
                    'perf_event_open',
                    'pipe2',
                    'readlinkat',
                    'remap_file_pages',
                    'sendfile',
                    'sendfile64',
                    'setns',
                    'shmat',
                    'shmctl',
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'unlink',
                    'unlinkat',
                    'uselib']),
    'futex': set(['clock_nanosleep',
                  'epoll_wait',
                  'faccessat',
                  'fadvise64_64',
                  'fchmod',
                  'fchmodat',
                  'fchown',
                  'fchownat',
                  'ftruncate',
                  'futimesat',
                  'inotify_add_watch',
                  'io_getevents',
                  'ioctl',
                  'linkat',
```

```
1446                'memfd_create',
1447                'mq_getsetattr',
1448                'mq_notify',
1449                'mq_timedreceive',
1450                'mq_timedsend',
1451                'mq_unlink',
1452                'nanosleep',
1453                'poll',
1454                'ppoll',
1455                'pselect6',
1456                'readlinkat',
1457                'recvmmsg',
1458                'rt_sigtimedwait',
1459                'sched_rr_get_interval',
1460                'select',
1461                'semtimedop',
1462                'sendfile',
1463                'sendfile64',
1464                'settimeofday',
1465                'stime',
1466                'swapoff',
1467                'swapon',
1468                'timer_gettime',
1469                'timer_settime',
1470                'timerfd_gettime',
1471                'timerfd_settime',
1472                'unlink',
1473                'unlinkat',
1474                'uselib',
1475                'utime']),
1476    'futimesat': set(['adjtimex',
1477                      'alarm',
1478                      'clock_adjtime',
1479                      'clock_nanosleep',
1480                      'epoll_wait',
1481                      'faccessat',
1482                      'fadvise64_64',
1483                      'fchmod',
1484                      'fchmodat',
1485                      'fchown',
1486                      'fchownat',
1487                      'ftruncate',
1488                      'futex',
1489                      'getitimer',
1490                      'getrusage',
1491                      'inotify_add_watch',
1492                      'io_getevents',
1493                      'ioctl',
1494                      'linkat',
1495                      'memfd_create',
1496                      'mq_getsetattr',
1497                      'mq_notify',
1498                      'mq_timedreceive',
1499                      'mq_timedsend',
1500                      'mq_unlink',
1501                      'nanosleep',
1502                      'poll',
1503                      'ppoll',
1504                      'pselect6',
1505                      'readlinkat',
1506                      'recvmmsg',
1507                      'rt_sigtimedwait',
1508                      'sched_rr_get_interval',
1509                      'select',
1510                      'semtimedop',
1511                      'sendfile',
```



```
1512                      'sendfile64',
1513                      'setitimer',
1514                      'settimeofday',
1515                      'stime',
1516                      'swapoff',
1517                      'swapon',
1518                      'timer_gettime',
1519                      'timer_settime',
1520                      'timerfd_gettime',
1521                      'timerfd_settime',
1522                      'unlink',
1523                      'unlinkat',
1524                      'uselib',
1525                      'utime',
1526                      'wait4',
1527                      'waitid']),
1528    'get_mempolicy': set(['capget',
1529                          'clone',
1530                          'fork',
1531                          'get_robust_list',
1532                          'getitimer',
1533                          'getpgid',
1534                          'getppid',
1535                          'getpriority',
1536                          'getrusage',
1537                          'getsid',
1538                          'ioprio_get',
1539                          'ioprio_set',
1540                          'keyctl',
1541                          'kill',
1542                          'mbind',
1543                          'migrate_pages',
1544                          'move_pages',
1545                          'mq_timedreceive',
1546                          'mq_timedsend',
1547                          'msgrcv',
1548                          'perf_event_open',
1549                          'prctl',
1550                          'prlimit64',
1551                          'ptrace',
1552                          'rt_sigaction',
1553                          'rt_sigprocmask',
1554                          'rt_sigtimedwait',
1555                          'sched_getaffinity',
1556                          'sched_getattr',
1557                          'sched_getparam',
1558                          'sched_getscheduler',
1559                          'sched_rr_get_interval',
1560                          'sched_setaffinity',
1561                          'sched_setattr',
1562                          'sched_setparam',
1563                          'sched_setscheduler',
1564                          'semtimedop',
1565                          'set_mempolicy',
1566                          'setitimer',
1567                          'setns',
1568                          'setpgid',
1569                          'setpriority',
1570                          'setsid',
1571                          'sigaction',
1572                          'sigaltstack',
1573                          'signal',
1574                          'umount',
1575                          'vfork']),
1576    'getdents': set(['accept4',
1577                     'bind',
```

```
                  'bpf',
                  'capget',
                  'clone',
                  'connect',
                  'copy_file_range',
                  'epoll_ctl',
                  'epoll_wait',
                  'fadvise64_64',
                  'fallocate',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
                  'fork',
                  'fremovexattr',
                  'fsetxattr',
                  'fstatfs',
                  'fstatfs64',
                  'fsync',
                  'ftruncate',
                  'futimesat',
                  'get_robust_list',
                  'getdents64',
                  'getitimer',
                  'getpeername',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'inotify_rm_watch',
                  'ioctl',
                  'ioperm',
                  'iopl',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'listen',
                  'llseek',
                  'lseek',
                  'migrate_pages',
                  'move_pages',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'msgrcv',
                  'old_readdir',
                  'perf_event_open',
                  'prctl',
                  'pread64',
                  'preadv',
                  'preadv2',
                  'preadv64',
                  'preadv64v2',
                  'prlimit64',
                  'ptrace',
```



```
                  'pwrite64',
                  'pwritev',
                  'pwritev2',
                  'pwritev64',
                  'pwritev64v2',
                  'read',
                  'readahead',
                  'readv',
                  'recvfrom',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'sendfile',
                  'sendfile64',
                  'sendto',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'setsockopt',
                  'shutdown',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'signalfd4',
                  'splice',
                  'sync_file_range',
                  'syncfs',
                  'tee',
                  'umount',
                  'utime',
                  'utimensat',
                  'vfork',
                  'vmsplice',
                  'write',
                  'writev']),
  'getdents64': set(['accept4',
                     'bind',
                     'bpf',
                     'capget',
                     'clone',
                     'connect',
                     'copy_file_range',
                     'epoll_ctl',
                     'epoll_wait',
                     'fadvise64_64',
                     'fallocate',
                     'fchdir',
                     'fchmod',
                     'fchown',
                     'fcntl',
                     'fcntl64',
                     'fdatasync',
                     'fgetxattr',
                     'flistxattr',
```

```
                    'flock',
                    'fork',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'get_robust_list',
                    'getdents',
                    'getitimer',
                    'getpeername',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'ioperm',
                    'iopl',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'listen',
                    'llseek',
                    'lseek',
                    'migrate_pages',
                    'move_pages',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'old_readdir',
                    'perf_event_open',
                    'prctl',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'prlimit64',
                    'ptrace',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
```



```
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'setsockopt',
                    'shutdown',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'umount',
                    'utime',
                    'utimensat',
                    'vfork',
                    'vmsplice',
                    'write',
                    'writev']),
    'getgroups16': set(['setgroups', 'setgroups16']),
    'getitimer': set(['exit_group', 'setitimer', 'timer_create']),
    'getrlimit': set(['old_getrlimit', 'prlimit64', 'setrlimit']),
    'getrusage': set(['exit_group', 'timer_create']),
    'getsockopt': set(['accept4']),
    'init_module': set(['delete_module', 'finit_module']),
    'inotify_add_watch': set(['accept4',
                              'bind',
                              'bpf',
                              'connect',
                              'copy_file_range',
                              'epoll_ctl',
                              'epoll_wait',
                              'fadvise64_64',
                              'fallocate',
                              'fchdir',
                              'fchmod',
                              'fchown',
                              'fcntl',
                              'fcntl64',
                              'fdatasync',
                              'fgetxattr',
                              'flistxattr',
                              'flock',
                              'fremovexattr',
                              'fsetxattr',
                              'fstatfs',
                              'fstatfs64',
                              'fsync',
                              'ftruncate',
                              'futimesat',
                              'getdents',
                              'getdents64',
                              'getpeername',
                              'getsockname',
                              'getsockopt',
```

```
                          'inotify_rm_watch',
                          'ioctl',
                          'listen',
                          'llseek',
                          'lseek',
                          'mq_getsetattr',
                          'mq_notify',
                          'mq_timedreceive',
                          'mq_timedsend',
                          'old_readdir',
                          'perf_event_open',
                          'pread64',
                          'preadv',
                          'preadv2',
                          'preadv64',
                          'preadv64v2',
                          'pwrite64',
                          'pwritev',
                          'pwritev2',
                          'pwritev64',
                          'pwritev64v2',
                          'read',
                          'readahead',
                          'readv',
                          'recvfrom',
                          'sendfile',
                          'sendfile64',
                          'sendto',
                          'setsockopt',
                          'shutdown',
                          'signalfd4',
                          'splice',
                          'sync_file_range',
                          'syncfs',
                          'tee',
                          'utime',
                          'utimensat',
                          'vmsplice',
                          'write',
                          'writev']),
 'inotify_rm_watch': set(['accept4',
                          'bind',
                          'bpf',
                          'connect',
                          'copy_file_range',
                          'epoll_ctl',
                          'epoll_wait',
                          'fadvise64_64',
                          'fallocate',
                          'fchdir',
                          'fchmod',
                          'fchown',
                          'fcntl',
                          'fcntl64',
                          'fdatasync',
                          'fgetxattr',
                          'flistxattr',
                          'flock',
                          'fremovexattr',
                          'fsetxattr',
                          'fstatfs',
                          'fstatfs64',
                          'fsync',
                          'ftruncate',
                          'futimesat',
                          'getdents',
```



```
                          'getdents64',
                          'getpeername',
                          'getsockname',
                          'getsockopt',
                          'inotify_add_watch',
                          'ioctl',
                          'listen',
                          'llseek',
                          'lseek',
                          'mq_getsetattr',
                          'mq_notify',
                          'mq_timedreceive',
                          'mq_timedsend',
                          'old_readdir',
                          'perf_event_open',
                          'pread64',
                          'preadv',
                          'preadv2',
                          'preadv64',
                          'preadv64v2',
                          'pwrite64',
                          'pwritev',
                          'pwritev2',
                          'pwritev64',
                          'pwritev64v2',
                          'read',
                          'readahead',
                          'readv',
                          'recvfrom',
                          'sendfile',
                          'sendfile64',
                          'sendto',
                          'setsockopt',
                          'shutdown',
                          'signalfd4',
                          'splice',
                          'sync_file_range',
                          'syncfs',
                          'tee',
                          'utime',
                          'utimensat',
                          'vmsplice',
                          'write',
                          'writev']),
 'io_cancel': set(['io_destroy', 'io_getevents', 'io_setup', 'io_submit']),
 'io_destroy': set(['io_cancel', 'io_getevents', 'io_setup', 'io_submit']),
 'io_getevents': set(['capget',
                      'clone',
                      'fork',
                      'get_robust_list',
                      'getitimer',
                      'getpgid',
                      'getppid',
                      'getpriority',
                      'getrusage',
                      'getsid',
                      'io_cancel',
                      'io_destroy',
                      'io_setup',
                      'io_submit',
                      'ioprio_get',
                      'ioprio_set',
                      'keyctl',
                      'kill',
                      'migrate_pages',
                      'move_pages',
```

```
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
 'io_setup': set(['io_cancel', 'io_destroy', 'io_getevents', 'io_submit']),
 'io_submit': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioperm',
                   'iopl',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
```



```
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaction',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'ioctl': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fadvise64_64',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
               'flistxattr',
               'flock',
               'fremovexattr',
               'fsetxattr',
               'fstatfs',
               'fstatfs64',
               'fsync',
               'ftruncate',
               'futimesat',
               'getdents',
               'getdents64',
               'getpeername',
               'getsockname',
               'getsockopt',
               'inotify_add_watch',
               'inotify_rm_watch',
               'listen',
               'llseek',
               'lseek',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'old_readdir',
               'perf_event_open',
               'pread64',
               'preadv',
               'preadv2',
               'preadv64',
               'preadv64v2',
               'pwrite64',
               'pwritev',
               'pwritev2',
               'pwritev64',
               'pwritev64v2',
               'read',
               'readahead',
```

```
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'syncfs',
                'tee',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'ioperm': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'iopl',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'keyctl': set(['capget',
                'clone',
```



```
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'request_key',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'llseek': set(['accept4',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'epoll_ctl',
                'epoll_wait',
                'fadvise64_64',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
```

```
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'lseek',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'perf_event_open',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'syncfs',
                'tee',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
    'lseek': set(['accept4',
                  'bind',
                  'bpf',
                  'connect',
                  'copy_file_range',
                  'epoll_ctl',
                  'epoll_wait',
                  'fadvise64_64',
                  'fallocate',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
```



```
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'old_readdir',
                'perf_event_open',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'sendfile',
                'sendfile64',
                'sendto',
                'setsockopt',
                'shutdown',
                'signalfd4',
                'splice',
                'sync_file_range',
                'syncfs',
                'tee',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
    'lstat': set(['fstat', 'stat']),
    'madvise': set(['brk',
                    'get_mempolicy',
                    'mincore',
                    'mlockall',
                    'mprotect',
                    'mremap',
                    'msync',
                    'munlock',
                    'munlockall',
                    'pkey_mprotect',
                    'prctl',
                    'remap_file_pages',
                    'shmdt']),
```

```
'migrate_pages': set(['capget',
                      'clone',
                      'fork',
                      'get_robust_list',
                      'getitimer',
                      'getpgid',
                      'getppid',
                      'getpriority',
                      'getrusage',
                      'getsid',
                      'ioperm',
                      'iopl',
                      'ioprio_get',
                      'ioprio_set',
                      'keyctl',
                      'kill',
                      'move_pages',
                      'mq_timedreceive',
                      'mq_timedsend',
                      'msgrcv',
                      'perf_event_open',
                      'prctl',
                      'prlimit64',
                      'ptrace',
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setsid',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'umount',
                      'vfork']),
'mincore': set(['brk',
                'capget',
                'clone',
                'fork',
                'get_mempolicy',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioperm',
                'iopl',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'madvise',
```



```
                'migrate_pages',
                'mlockall',
                'move_pages',
                'mprotect',
                'mq_timedreceive',
                'mq_timedsend',
                'mremap',
                'msgrcv',
                'msync',
                'munlock',
                'munlockall',
                'perf_event_open',
                'pkey_mprotect',
                'prctl',
                'prlimit64',
                'ptrace',
                'remap_file_pages',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmdt',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
'mkdirat': set(['fadvise64_64',
                'quotactl',
                'swapon',
                'syncfs',
                'umount',
                'ustat']),
'mknodat': set(['fadvise64_64',
                'quotactl',
                'swapon',
                'syncfs',
                'umount',
                'ustat']),
'mlockall': set(['brk',
                 'capget',
                 'clone',
                 'fork',
                 'get_mempolicy',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'io_cancel',
```

```
                  'io_destroy',
                  'io_getevents',
                  'io_setup',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'madvise',
                  'mbind',
                  'migrate_pages',
                  'mincore',
                  'modify_ldt',
                  'move_pages',
                  'mprotect',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mremap',
                  'msgrcv',
                  'msync',
                  'munlock',
                  'munlockall',
                  'perf_event_open',
                  'personality',
                  'pkey_mprotect',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'remap_file_pages',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'shmdt',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'swapoff',
                  'umount',
                  'vfork']),
  'modify_ldt': set(['get_thread_area', 'set_thread_area']),
  'mount': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
```



```
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'personality',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
  'mprotect': set(['brk',
                   'capget',
                   'clone',
                   'fork',
                   'get_mempolicy',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'madvise',
                   'migrate_pages',
                   'mincore',
                   'mlockall',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'mremap',
                   'msgrcv',
                   'msync',
                   'munlock',
                   'munlockall',
                   'perf_event_open',
                   'personality',
                   'pkey_mprotect',
```

```
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'remap_file_pages',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'semtimedop',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'shmdt',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'umount',
                  'vfork']),
 'mq_getsetattr': set(['mq_notify',
                       'mq_open',
                       'mq_timedreceive',
                       'mq_timedsend']),
 'mq_notify': set(['mq_getsetattr',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'rt_sigqueueinfo',
                   'rt_sigreturn',
                   'rt_sigtimedwait',
                   'rt_tgsigqueueinfo',
                   'tgkill',
                   'timer_create',
                   'tkill']),
 'mq_open': set(['acct',
                 'mq_unlink',
                 'open',
                 'openat',
                 'quotactl',
                 'renameat2',
                 'rmdir',
                 'swapoff',
                 'swapon',
                 'symlinkat',
                 'sysfs',
                 'unlink',
                 'unlinkat',
                 'uselib']),
 'mq_timedreceive': set(['accept4',
                         'acct',
                         'dup',
                         'dup3',
                         'epoll_create1',
                         'epoll_ctl',
                         'eventfd2',
                         'fadvise64_64',
                         'flock',
                         'memfd_create',
```



```
                         'mmap_pgoff',
                         'mq_getsetattr',
                         'mq_notify',
                         'mq_open',
                         'mq_timedsend',
                         'msgrcv',
                         'msgsnd',
                         'msync',
                         'open',
                         'open_by_handle_at',
                         'openat',
                         'perf_event_open',
                         'pipe2',
                         'remap_file_pages',
                         'setns',
                         'shmat',
                         'shmctl',
                         'shmdt',
                         'socket',
                         'socketpair',
                         'swapoff',
                         'swapon',
                         'uselib']),
 'mq_timedsend': set(['accept4',
                      'acct',
                      'dup',
                      'dup3',
                      'epoll_create1',
                      'epoll_ctl',
                      'eventfd2',
                      'fadvise64_64',
                      'flock',
                      'memfd_create',
                      'mmap_pgoff',
                      'mq_getsetattr',
                      'mq_notify',
                      'mq_open',
                      'mq_timedreceive',
                      'msync',
                      'open',
                      'open_by_handle_at',
                      'openat',
                      'perf_event_open',
                      'pipe2',
                      'remap_file_pages',
                      'setns',
                      'shmat',
                      'shmctl',
                      'shmdt',
                      'socket',
                      'socketpair',
                      'swapoff',
                      'swapon',
                      'uselib']),
 'mremap': set(['brk',
                'capget',
                'clone',
                'fork',
                'get_mempolicy',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
```

```
                'io_cancel',
                'io_destroy',
                'io_getevents',
                'io_setup',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'madvise',
                'mbind',
                'migrate_pages',
                'mincore',
                'mlockall',
                'modify_ldt',
                'move_pages',
                'mprotect',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'msync',
                'munlock',
                'munlockall',
                'perf_event_open',
                'personality',
                'pkey_mprotect',
                'prctl',
                'prlimit64',
                'ptrace',
                'remap_file_pages',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmdt',
                'sigaction',
                'sigaltstack',
                'signal',
                'swapoff',
                'umount',
                'vfork']),
  'msgctl': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioperm',
                'iopl',
```



```
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'msgget',
                'msgrcv',
                'msgsnd',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semctl',
                'semget',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmat',
                'shmctl',
                'shmget',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
  'msgrcv': set(['mq_timedreceive', 'mq_timedsend', 'msgsnd']),
  'msgsnd': set(['mq_open',
                'mq_unlink',
                'msgctl',
                'msgget',
                'msgrcv',
                'semctl',
                'semget',
                'semtimedop',
                'setns',
                'shmat',
                'shmctl',
                'shmget']),
  'msync': set(['brk',
                'get_mempolicy',
                'madvise',
                'mincore',
                'mlockall',
                'mprotect',
                'mremap',
                'munlock',
```

```
2898                'munlockall',
2899                'pkey_mprotect',
2900                'prctl',
2901                'remap_file_pages',
2902                'shmdt']),
2903  'munlock': set(['brk',
2904                  'get_mempolicy',
2905                  'madvise',
2906                  'mincore',
2907                  'mlockall',
2908                  'mprotect',
2909                  'mremap',
2910                  'msync',
2911                  'munlockall',
2912                  'pkey_mprotect',
2913                  'prctl',
2914                  'remap_file_pages',
2915                  'shmdt']),
2916  'munlockall': set(['brk',
2917                     'get_mempolicy',
2918                     'madvise',
2919                     'mincore',
2920                     'mlockall',
2921                     'mprotect',
2922                     'mremap',
2923                     'msync',
2924                     'munlock',
2925                     'pkey_mprotect',
2926                     'prctl',
2927                     'remap_file_pages',
2928                     'shmdt']),
2929  'nanosleep': set(['clock_nanosleep',
2930                    'epoll_wait',
2931                    'faccessat',
2932                    'fadvise64_64',
2933                    'fchmod',
2934                    'fchmodat',
2935                    'fchown',
2936                    'fchownat',
2937                    'ftruncate',
2938                    'futex',
2939                    'futimesat',
2940                    'inotify_add_watch',
2941                    'io_getevents',
2942                    'ioctl',
2943                    'linkat',
2944                    'memfd_create',
2945                    'mq_getsetattr',
2946                    'mq_notify',
2947                    'mq_timedreceive',
2948                    'mq_timedsend',
2949                    'mq_unlink',
2950                    'poll',
2951                    'ppoll',
2952                    'pselect6',
2953                    'readlinkat',
2954                    'recvmmsg',
2955                    'rt_sigtimedwait',
2956                    'sched_rr_get_interval',
2957                    'select',
2958                    'semtimedop',
2959                    'sendfile',
2960                    'sendfile64',
2961                    'settimeofday',
2962                    'stime',
2963                    'swapoff',
```



```
2964                    'swapon',
2965                    'timer_gettime',
2966                    'timer_settime',
2967                    'timerfd_gettime',
2968                    'timerfd_settime',
2969                    'unlink',
2970                    'unlinkat',
2971                    'uselib',
2972                    'utime']),
2973  'newfstat': set(['newfstatat', 'newlstat', 'newstat']),
2974  'newfstatat': set(['newfstat', 'newlstat', 'newstat']),
2975  'newlstat': set(['newfstat', 'newfstatat', 'newstat']),
2976  'newstat': set(['newfstat', 'newfstatat', 'newlstat']),
2977  'newuname': set(['capget',
2978                   'clone',
2979                   'fork',
2980                   'get_robust_list',
2981                   'getitimer',
2982                   'getpgid',
2983                   'getppid',
2984                   'getpriority',
2985                   'getrusage',
2986                   'getsid',
2987                   'ioprio_get',
2988                   'ioprio_set',
2989                   'keyctl',
2990                   'kill',
2991                   'migrate_pages',
2992                   'move_pages',
2993                   'mq_timedreceive',
2994                   'mq_timedsend',
2995                   'msgrcv',
2996                   'perf_event_open',
2997                   'personality',
2998                   'prctl',
2999                   'prlimit64',
3000                   'ptrace',
3001                   'rt_sigaction',
3002                   'rt_sigprocmask',
3003                   'rt_sigtimedwait',
3004                   'sched_getaffinity',
3005                   'sched_getattr',
3006                   'sched_getparam',
3007                   'sched_getscheduler',
3008                   'sched_rr_get_interval',
3009                   'sched_setaffinity',
3010                   'sched_setattr',
3011                   'sched_setparam',
3012                   'sched_setscheduler',
3013                   'semtimedop',
3014                   'setitimer',
3015                   'setns',
3016                   'setpgid',
3017                   'setpriority',
3018                   'setsid',
3019                   'sigaction',
3020                   'sigaltstack',
3021                   'signal',
3022                   'umount',
3023                   'vfork']),
3024  'old_getrlimit': set(['prlimit64', 'setrlimit']),
3025  'old_readdir': set(['accept4',
3026                      'bind',
3027                      'bpf',
3028                      'connect',
3029                      'copy_file_range',
```

```
                    'epoll_ctl',
                    'epoll_wait',
                    'fadvise64_64',
                    'fallocate',
                    'fchdir',
                    'fchmod',
                    'fchown',
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'signalfd4',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
'olduname': set(['capget',
```



```
                    'clone',
                    'fork',
                    'get_robust_list',
                    'getitimer',
                    'getpgid',
                    'getppid',
                    'getpriority',
                    'getrusage',
                    'getsid',
                    'ioperm',
                    'iopl',
                    'ioprio_get',
                    'ioprio_set',
                    'keyctl',
                    'kill',
                    'migrate_pages',
                    'move_pages',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'msgrcv',
                    'perf_event_open',
                    'personality',
                    'prctl',
                    'prlimit64',
                    'ptrace',
                    'rt_sigaction',
                    'rt_sigprocmask',
                    'rt_sigtimedwait',
                    'sched_getaffinity',
                    'sched_getattr',
                    'sched_getparam',
                    'sched_getscheduler',
                    'sched_rr_get_interval',
                    'sched_setaffinity',
                    'sched_setattr',
                    'sched_setparam',
                    'sched_setscheduler',
                    'semtimedop',
                    'setitimer',
                    'setns',
                    'setpgid',
                    'setpriority',
                    'setsid',
                    'sigaction',
                    'sigaltstack',
                    'signal',
                    'umount',
                    'vfork']),
'perf_event_open': set(['accept4',
                        'bind',
                        'bpf',
                        'capget',
                        'clone',
                        'connect',
                        'copy_file_range',
                        'epoll_ctl',
                        'epoll_wait',
                        'fadvise64_64',
                        'fallocate',
                        'fchdir',
                        'fchmod',
                        'fchown',
                        'fcntl',
                        'fcntl64',
                        'fdatasync',
                        'fgetxattr',
```

```
                        'flistxattr',
                        'flock',
                        'fork',
                        'fremovexattr',
                        'fsetxattr',
                        'fstatfs',
                        'fstatfs64',
                        'fsync',
                        'ftruncate',
                        'futimesat',
                        'get_robust_list',
                        'getdents',
                        'getdents64',
                        'getitimer',
                        'getpeername',
                        'getpgid',
                        'getppid',
                        'getpriority',
                        'getrusage',
                        'getsid',
                        'getsockname',
                        'getsockopt',
                        'inotify_add_watch',
                        'inotify_rm_watch',
                        'ioctl',
                        'ioperm',
                        'iopl',
                        'ioprio_get',
                        'ioprio_set',
                        'keyctl',
                        'kill',
                        'listen',
                        'llseek',
                        'lseek',
                        'migrate_pages',
                        'move_pages',
                        'mq_getsetattr',
                        'mq_notify',
                        'mq_timedreceive',
                        'mq_timedsend',
                        'msgrcv',
                        'old_readdir',
                        'prctl',
                        'pread64',
                        'preadv',
                        'preadv2',
                        'preadv64',
                        'preadv64v2',
                        'prlimit64',
                        'ptrace',
                        'pwrite64',
                        'pwritev',
                        'pwritev2',
                        'pwritev64',
                        'pwritev64v2',
                        'read',
                        'readahead',
                        'readv',
                        'recvfrom',
                        'rt_sigaction',
                        'rt_sigprocmask',
                        'rt_sigtimedwait',
                        'sched_getaffinity',
                        'sched_getattr',
                        'sched_getparam',
                        'sched_getscheduler',
```



```
                        'sched_rr_get_interval',
                        'sched_setaffinity',
                        'sched_setattr',
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'sendfile',
                        'sendfile64',
                        'sendto',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setresuid',
                        'setreuid',
                        'setsid',
                        'setsockopt',
                        'setuid',
                        'shutdown',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'signalfd4',
                        'splice',
                        'sync_file_range',
                        'syncfs',
                        'tee',
                        'umount',
                        'utime',
                        'utimensat',
                        'vfork',
                        'vmsplice',
                        'write',
                        'writev']),
    'pivot_root': set(['acct', 'getcwd', 'mq_open', 'mq_unlink', 'umount']),
    'pkey_alloc': set(['brk',
                       'get_mempolicy',
                       'getrusage',
                       'io_cancel',
                       'io_destroy',
                       'io_getevents',
                       'io_setup',
                       'mbind',
                       'migrate_pages',
                       'mincore',
                       'modify_ldt',
                       'move_pages',
                       'mremap',
                       'msync',
                       'pkey_free',
                       'prctl',
                       'remap_file_pages',
                       'shmdt',
                       'swapoff']),
    'pkey_mprotect': set(['brk',
                          'capget',
                          'clone',
                          'fork',
                          'get_mempolicy',
                          'get_robust_list',
                          'getitimer',
                          'getpgid',
                          'getppid',
                          'getpriority',
                          'getrusage',
                          'getsid',
```

```
                        'ioprio_get',
                        'ioprio_set',
                        'keyctl',
                        'kill',
                        'madvise',
                        'migrate_pages',
                        'mincore',
                        'mlockall',
                        'move_pages',
                        'mprotect',
                        'mq_timedreceive',
                        'mq_timedsend',
                        'mremap',
                        'msgrcv',
                        'msync',
                        'munlock',
                        'munlockall',
                        'perf_event_open',
                        'personality',
                        'prctl',
                        'prlimit64',
                        'ptrace',
                        'remap_file_pages',
                        'rt_sigaction',
                        'rt_sigprocmask',
                        'rt_sigtimedwait',
                        'sched_getaffinity',
                        'sched_getattr',
                        'sched_getparam',
                        'sched_getscheduler',
                        'sched_rr_get_interval',
                        'sched_setaffinity',
                        'sched_setattr',
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setsid',
                        'shmdt',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'umount',
                        'vfork']),
    'poll': set(['ppoll']),
    'ppoll': set(['capget',
                  'clock_nanosleep',
                  'clone',
                  'epoll_wait',
                  'faccessat',
                  'fadvise64_64',
                  'fchmod',
                  'fchmodat',
                  'fchown',
                  'fchownat',
                  'fork',
                  'ftruncate',
                  'futex',
                  'futimesat',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
```



```
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'inotify_add_watch',
                  'io_getevents',
                  'ioctl',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'linkat',
                  'memfd_create',
                  'migrate_pages',
                  'move_pages',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mq_unlink',
                  'msgrcv',
                  'nanosleep',
                  'perf_event_open',
                  'personality',
                  'poll',
                  'prctl',
                  'prlimit64',
                  'pselect6',
                  'ptrace',
                  'readlinkat',
                  'recvmmsg',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
                  'sched_setparam',
                  'sched_setscheduler',
                  'select',
                  'semtimedop',
                  'sendfile',
                  'sendfile64',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'settimeofday',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'stime',
                  'swapoff',
                  'swapon',
                  'timer_gettime',
                  'timer_settime',
                  'timerfd_gettime',
                  'timerfd_settime',
                  'umount',
                  'unlink',
                  'unlinkat',
                  'uselib',
```

```
3426                 'utime',
3427                 'vfork']),
3428  'prctl': set(['brk',
3429                 'capget',
3430                 'clone',
3431                 'fork',
3432                 'get_mempolicy',
3433                 'get_robust_list',
3434                 'getitimer',
3435                 'getpgid',
3436                 'getppid',
3437                 'getpriority',
3438                 'getrusage',
3439                 'getsid',
3440                 'io_cancel',
3441                 'io_destroy',
3442                 'io_getevents',
3443                 'io_setup',
3444                 'ioprio_get',
3445                 'ioprio_set',
3446                 'keyctl',
3447                 'kill',
3448                 'mbind',
3449                 'migrate_pages',
3450                 'mincore',
3451                 'modify_ldt',
3452                 'move_pages',
3453                 'mq_timedreceive',
3454                 'mq_timedsend',
3455                 'mremap',
3456                 'msgrcv',
3457                 'msync',
3458                 'perf_event_open',
3459                 'personality',
3460                 'prlimit64',
3461                 'ptrace',
3462                 'remap_file_pages',
3463                 'rt_sigaction',
3464                 'rt_sigprocmask',
3465                 'rt_sigtimedwait',
3466                 'sched_getaffinity',
3467                 'sched_getattr',
3468                 'sched_getparam',
3469                 'sched_getscheduler',
3470                 'sched_rr_get_interval',
3471                 'sched_setaffinity',
3472                 'sched_setattr',
3473                 'sched_setparam',
3474                 'sched_setscheduler',
3475                 'semtimedop',
3476                 'setitimer',
3477                 'setns',
3478                 'setpgid',
3479                 'setpriority',
3480                 'setresuid',
3481                 'setreuid',
3482                 'setsid',
3483                 'setuid',
3484                 'shmdt',
3485                 'sigaction',
3486                 'sigaltstack',
3487                 'signal',
3488                 'swapoff',
3489                 'umount',
3490                 'vfork']),
3491  'pread64': set(['accept4',
```



```
3492                 'acct',
3493                 'bind',
3494                 'bpf',
3495                 'connect',
3496                 'copy_file_range',
3497                 'dup',
3498                 'dup3',
3499                 'epoll_create1',
3500                 'epoll_ctl',
3501                 'epoll_wait',
3502                 'eventfd2',
3503                 'fadvise64_64',
3504                 'fallocate',
3505                 'fchdir',
3506                 'fchmod',
3507                 'fchown',
3508                 'fcntl',
3509                 'fcntl64',
3510                 'fdatasync',
3511                 'fgetxattr',
3512                 'flistxattr',
3513                 'flock',
3514                 'fremovexattr',
3515                 'fsetxattr',
3516                 'fstatfs',
3517                 'fstatfs64',
3518                 'fsync',
3519                 'ftruncate',
3520                 'futimesat',
3521                 'getdents',
3522                 'getdents64',
3523                 'getpeername',
3524                 'getsockname',
3525                 'getsockopt',
3526                 'inotify_add_watch',
3527                 'inotify_rm_watch',
3528                 'ioctl',
3529                 'listen',
3530                 'llseek',
3531                 'lseek',
3532                 'memfd_create',
3533                 'mmap_pgoff',
3534                 'mq_getsetattr',
3535                 'mq_notify',
3536                 'mq_open',
3537                 'mq_timedreceive',
3538                 'mq_timedsend',
3539                 'msync',
3540                 'old_readdir',
3541                 'open',
3542                 'open_by_handle_at',
3543                 'openat',
3544                 'perf_event_open',
3545                 'pipe2',
3546                 'preadv',
3547                 'preadv2',
3548                 'preadv64',
3549                 'preadv64v2',
3550                 'pwrite64',
3551                 'pwritev',
3552                 'pwritev2',
3553                 'pwritev64',
3554                 'pwritev64v2',
3555                 'read',
3556                 'readahead',
3557                 'readv',
```

```
3558                 'recvfrom',
3559                 'remap_file_pages',
3560                 'sendfile',
3561                 'sendfile64',
3562                 'sendto',
3563                 'setns',
3564                 'setsockopt',
3565                 'shmat',
3566                 'shmctl',
3567                 'shmdt',
3568                 'shutdown',
3569                 'signalfd4',
3570                 'socket',
3571                 'socketpair',
3572                 'splice',
3573                 'swapoff',
3574                 'swapon',
3575                 'sync_file_range',
3576                 'syncfs',
3577                 'tee',
3578                 'uselib',
3579                 'utime',
3580                 'utimensat',
3581                 'vmsplice',
3582                 'write',
3583                 'writev']),
3584  'preadv': set(['accept4',
3585                 'acct',
3586                 'dup',
3587                 'dup3',
3588                 'epoll_create1',
3589                 'epoll_ctl',
3590                 'eventfd2',
3591                 'fadvise64_64',
3592                 'flock',
3593                 'memfd_create',
3594                 'mmap_pgoff',
3595                 'mq_open',
3596                 'msync',
3597                 'open',
3598                 'open_by_handle_at',
3599                 'openat',
3600                 'perf_event_open',
3601                 'pipe2',
3602                 'remap_file_pages',
3603                 'setns',
3604                 'shmat',
3605                 'shmctl',
3606                 'shmdt',
3607                 'socket',
3608                 'socketpair',
3609                 'swapoff',
3610                 'swapon',
3611                 'uselib']),
3612  'preadv2': set(['accept4',
3613                  'acct',
3614                  'dup',
3615                  'dup3',
3616                  'epoll_create1',
3617                  'epoll_ctl',
3618                  'eventfd2',
3619                  'fadvise64_64',
3620                  'flock',
3621                  'memfd_create',
3622                  'mmap_pgoff',
3623                  'mq_open',
```



```
3624                  'msync',
3625                  'open',
3626                  'open_by_handle_at',
3627                  'openat',
3628                  'perf_event_open',
3629                  'pipe2',
3630                  'remap_file_pages',
3631                  'setns',
3632                  'shmat',
3633                  'shmctl',
3634                  'shmdt',
3635                  'socket',
3636                  'socketpair',
3637                  'swapoff',
3638                  'swapon',
3639                  'uselib']),
3640  'preadv64': set(['accept4',
3641                   'acct',
3642                   'dup',
3643                   'dup3',
3644                   'epoll_create1',
3645                   'epoll_ctl',
3646                   'eventfd2',
3647                   'fadvise64_64',
3648                   'flock',
3649                   'memfd_create',
3650                   'mmap_pgoff',
3651                   'mq_open',
3652                   'msync',
3653                   'open',
3654                   'open_by_handle_at',
3655                   'openat',
3656                   'perf_event_open',
3657                   'pipe2',
3658                   'remap_file_pages',
3659                   'setns',
3660                   'shmat',
3661                   'shmctl',
3662                   'shmdt',
3663                   'socket',
3664                   'socketpair',
3665                   'swapoff',
3666                   'swapon',
3667                   'uselib']),
3668  'preadv64v2': set(['accept4',
3669                     'acct',
3670                     'dup',
3671                     'dup3',
3672                     'epoll_create1',
3673                     'epoll_ctl',
3674                     'eventfd2',
3675                     'fadvise64_64',
3676                     'flock',
3677                     'memfd_create',
3678                     'mmap_pgoff',
3679                     'mq_open',
3680                     'msync',
3681                     'open',
3682                     'open_by_handle_at',
3683                     'openat',
3684                     'perf_event_open',
3685                     'pipe2',
3686                     'remap_file_pages',
3687                     'setns',
3688                     'shmat',
3689                     'shmctl',
```

```
                    'shmdt',
                    'socket',
                    'socketpair',
                    'swapoff',
                    'swapon',
                    'uselib']),
 'prlimit64': set(['old_getrlimit', 'setrlimit']),
 'pselect6': set(['capget',
                  'clock_nanosleep',
                  'clone',
                  'epoll_wait',
                  'faccessat',
                  'fadvise64_64',
                  'fchmod',
                  'fchmodat',
                  'fchown',
                  'fchownat',
                  'fork',
                  'ftruncate',
                  'futex',
                  'futimesat',
                  'get_robust_list',
                  'getitimer',
                  'getpgid',
                  'getppid',
                  'getpriority',
                  'getrusage',
                  'getsid',
                  'inotify_add_watch',
                  'io_getevents',
                  'ioctl',
                  'ioperm',
                  'iopl',
                  'ioprio_get',
                  'ioprio_set',
                  'keyctl',
                  'kill',
                  'linkat',
                  'memfd_create',
                  'migrate_pages',
                  'move_pages',
                  'mq_getsetattr',
                  'mq_notify',
                  'mq_timedreceive',
                  'mq_timedsend',
                  'mq_unlink',
                  'msgrcv',
                  'nanosleep',
                  'perf_event_open',
                  'poll',
                  'ppoll',
                  'prctl',
                  'prlimit64',
                  'ptrace',
                  'readlinkat',
                  'recvmmsg',
                  'rt_sigaction',
                  'rt_sigprocmask',
                  'rt_sigtimedwait',
                  'sched_getaffinity',
                  'sched_getattr',
                  'sched_getparam',
                  'sched_getscheduler',
                  'sched_rr_get_interval',
                  'sched_setaffinity',
                  'sched_setattr',
```



```
                  'sched_setparam',
                  'sched_setscheduler',
                  'select',
                  'semtimedop',
                  'sendfile',
                  'sendfile64',
                  'setitimer',
                  'setns',
                  'setpgid',
                  'setpriority',
                  'setsid',
                  'settimeofday',
                  'sigaction',
                  'sigaltstack',
                  'signal',
                  'stime',
                  'swapoff',
                  'swapon',
                  'timer_gettime',
                  'timer_settime',
                  'timerfd_gettime',
                  'timerfd_settime',
                  'umount',
                  'unlink',
                  'unlinkat',
                  'uselib',
                  'utime',
                  'vfork']),
 'ptrace': set(['capget',
                'clone',
                'epoll_wait',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'pause',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigsuspend',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
```

```
3822                 'setitimer',
3823                 'setns',
3824                 'setpgid',
3825                 'setpriority',
3826                 'setresuid',
3827                 'setreuid',
3828                 'setsid',
3829                 'setuid',
3830                 'sigaction',
3831                 'sigaltstack',
3832                 'signal',
3833                 'sigsuspend',
3834                 'umount',
3835                 'vfork']),
3836  'pwrite64': set(['accept4',
3837                   'acct',
3838                   'bind',
3839                   'bpf',
3840                   'connect',
3841                   'copy_file_range',
3842                   'dup',
3843                   'dup3',
3844                   'epoll_create1',
3845                   'epoll_ctl',
3846                   'epoll_wait',
3847                   'eventfd2',
3848                   'fadvise64_64',
3849                   'fallocate',
3850                   'fchdir',
3851                   'fchmod',
3852                   'fchown',
3853                   'fcntl',
3854                   'fcntl64',
3855                   'fdatasync',
3856                   'fgetxattr',
3857                   'flistxattr',
3858                   'flock',
3859                   'fremovexattr',
3860                   'fsetxattr',
3861                   'fstatfs',
3862                   'fstatfs64',
3863                   'fsync',
3864                   'ftruncate',
3865                   'futimesat',
3866                   'getdents',
3867                   'getdents64',
3868                   'getpeername',
3869                   'getsockname',
3870                   'getsockopt',
3871                   'inotify_add_watch',
3872                   'inotify_rm_watch',
3873                   'ioctl',
3874                   'listen',
3875                   'llseek',
3876                   'lseek',
3877                   'memfd_create',
3878                   'mmap_pgoff',
3879                   'mq_getsetattr',
3880                   'mq_notify',
3881                   'mq_open',
3882                   'mq_timedreceive',
3883                   'mq_timedsend',
3884                   'msync',
3885                   'old_readdir',
3886                   'open',
3887                   'open_by_handle_at',
```



```
3888                   'openat',
3889                   'perf_event_open',
3890                   'pipe2',
3891                   'pread64',
3892                   'preadv',
3893                   'preadv2',
3894                   'preadv64',
3895                   'preadv64v2',
3896                   'pwritev',
3897                   'pwritev2',
3898                   'pwritev64',
3899                   'pwritev64v2',
3900                   'read',
3901                   'readahead',
3902                   'readv',
3903                   'recvfrom',
3904                   'remap_file_pages',
3905                   'sendfile',
3906                   'sendfile64',
3907                   'sendto',
3908                   'setns',
3909                   'setsockopt',
3910                   'shmat',
3911                   'shmctl',
3912                   'shmdt',
3913                   'shutdown',
3914                   'signalfd4',
3915                   'socket',
3916                   'socketpair',
3917                   'splice',
3918                   'swapoff',
3919                   'swapon',
3920                   'sync_file_range',
3921                   'syncfs',
3922                   'tee',
3923                   'uselib',
3924                   'utime',
3925                   'utimensat',
3926                   'vmsplice',
3927                   'write',
3928                   'writev']),
3929  'pwritev': set(['accept4',
3930                  'acct',
3931                  'dup',
3932                  'dup3',
3933                  'epoll_create1',
3934                  'epoll_ctl',
3935                  'eventfd2',
3936                  'fadvise64_64',
3937                  'flock',
3938                  'memfd_create',
3939                  'mmap_pgoff',
3940                  'mq_open',
3941                  'msync',
3942                  'open',
3943                  'open_by_handle_at',
3944                  'openat',
3945                  'perf_event_open',
3946                  'pipe2',
3947                  'remap_file_pages',
3948                  'setns',
3949                  'shmat',
3950                  'shmctl',
3951                  'shmdt',
3952                  'socket',
3953                  'socketpair',
```

```
                'swapoff',
                'swapon',
                'uselib']),
 'pwritev2': set(['accept4',
                  'acct',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'eventfd2',
                  'fadvise64_64',
                  'flock',
                  'memfd_create',
                  'mmap_pgoff',
                  'mq_open',
                  'msync',
                  'open',
                  'open_by_handle_at',
                  'openat',
                  'perf_event_open',
                  'pipe2',
                  'remap_file_pages',
                  'setns',
                  'shmat',
                  'shmctl',
                  'shmdt',
                  'socket',
                  'socketpair',
                  'swapoff',
                  'swapon',
                  'uselib']),
 'pwritev64': set(['accept4',
                   'acct',
                   'dup',
                   'dup3',
                   'epoll_create1',
                   'epoll_ctl',
                   'eventfd2',
                   'fadvise64_64',
                   'flock',
                   'memfd_create',
                   'mmap_pgoff',
                   'mq_open',
                   'msync',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'remap_file_pages',
                   'setns',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'socket',
                   'socketpair',
                   'swapoff',
                   'swapon',
                   'uselib']),
 'pwritev64v2': set(['accept4',
                     'acct',
                     'dup',
                     'dup3',
                     'epoll_create1',
                     'epoll_ctl',
                     'eventfd2',
```



```
                     'fadvise64_64',
                     'flock',
                     'memfd_create',
                     'mmap_pgoff',
                     'mq_open',
                     'msync',
                     'open',
                     'open_by_handle_at',
                     'openat',
                     'perf_event_open',
                     'pipe2',
                     'remap_file_pages',
                     'setns',
                     'shmat',
                     'shmctl',
                     'shmdt',
                     'socket',
                     'socketpair',
                     'swapoff',
                     'swapon',
                     'uselib']),
 'quotactl': set(['acct',
                  'fadvise64_64',
                  'mq_open',
                  'mq_unlink',
                  'open',
                  'openat',
                  'renameat2',
                  'rmdir',
                  'swapoff',
                  'swapon',
                  'symlinkat',
                  'syncfs',
                  'sysfs',
                  'umount',
                  'unlink',
                  'unlinkat',
                  'uselib',
                  'ustat']),
 'read': set(['accept4',
              'bind',
              'bpf',
              'connect',
              'copy_file_range',
              'epoll_ctl',
              'epoll_wait',
              'fadvise64_64',
              'fallocate',
              'fchdir',
              'fchmod',
              'fchown',
              'fcntl',
              'fcntl64',
              'fdatasync',
              'fgetxattr',
              'flistxattr',
              'flock',
              'fremovexattr',
              'fsetxattr',
              'fstatfs',
              'fstatfs64',
              'fsync',
              'ftruncate',
              'futimesat',
              'getdents',
              'getdents64',
```

```
4086                'getpeername',
4087                'getsockname',
4088                'getsockopt',
4089                'inotify_add_watch',
4090                'inotify_rm_watch',
4091                'ioctl',
4092                'listen',
4093                'llseek',
4094                'lseek',
4095                'mq_getsetattr',
4096                'mq_notify',
4097                'mq_timedreceive',
4098                'mq_timedsend',
4099                'old_readdir',
4100                'perf_event_open',
4101                'pread64',
4102                'preadv',
4103                'preadv2',
4104                'preadv64',
4105                'preadv64v2',
4106                'pwrite64',
4107                'pwritev',
4108                'pwritev2',
4109                'pwritev64',
4110                'pwritev64v2',
4111                'readahead',
4112                'readv',
4113                'recvfrom',
4114                'sendfile',
4115                'sendfile64',
4116                'sendto',
4117                'setsockopt',
4118                'shutdown',
4119                'signalfd4',
4120                'splice',
4121                'sync_file_range',
4122                'syncfs',
4123                'tee',
4124                'utime',
4125                'utimensat',
4126                'vmsplice',
4127                'write',
4128                'writev']),
4129  'readahead': set(['accept4',
4130                    'acct',
4131                    'bind',
4132                    'bpf',
4133                    'connect',
4134                    'copy_file_range',
4135                    'dup',
4136                    'dup3',
4137                    'epoll_create1',
4138                    'epoll_ctl',
4139                    'epoll_wait',
4140                    'eventfd2',
4141                    'fadvise64_64',
4142                    'fallocate',
4143                    'fchdir',
4144                    'fchmod',
4145                    'fchown',
4146                    'fcntl',
4147                    'fcntl64',
4148                    'fdatasync',
4149                    'fgetxattr',
4150                    'flistxattr',
4151                    'flock',
```



```
4152                    'fremovexattr',
4153                    'fsetxattr',
4154                    'fstatfs',
4155                    'fstatfs64',
4156                    'fsync',
4157                    'ftruncate',
4158                    'futimesat',
4159                    'getdents',
4160                    'getdents64',
4161                    'getpeername',
4162                    'getsockname',
4163                    'getsockopt',
4164                    'inotify_add_watch',
4165                    'inotify_rm_watch',
4166                    'ioctl',
4167                    'listen',
4168                    'llseek',
4169                    'lseek',
4170                    'memfd_create',
4171                    'mmap_pgoff',
4172                    'mq_getsetattr',
4173                    'mq_notify',
4174                    'mq_open',
4175                    'mq_timedreceive',
4176                    'mq_timedsend',
4177                    'msync',
4178                    'old_readdir',
4179                    'open',
4180                    'open_by_handle_at',
4181                    'openat',
4182                    'perf_event_open',
4183                    'pipe2',
4184                    'pread64',
4185                    'preadv',
4186                    'preadv2',
4187                    'preadv64',
4188                    'preadv64v2',
4189                    'pwrite64',
4190                    'pwritev',
4191                    'pwritev2',
4192                    'pwritev64',
4193                    'pwritev64v2',
4194                    'read',
4195                    'readv',
4196                    'recvfrom',
4197                    'remap_file_pages',
4198                    'sendfile',
4199                    'sendfile64',
4200                    'sendto',
4201                    'setns',
4202                    'setsockopt',
4203                    'shmat',
4204                    'shmctl',
4205                    'shmdt',
4206                    'shutdown',
4207                    'signalfd4',
4208                    'socket',
4209                    'socketpair',
4210                    'splice',
4211                    'swapoff',
4212                    'swapon',
4213                    'sync_file_range',
4214                    'syncfs',
4215                    'tee',
4216                    'uselib',
4217                    'utime',
```

```
4218                  'utimensat',
4219                  'vmsplice',
4220                  'write',
4221                  'writev']),
4222  'recvfrom': set(['accept4',
4223                   'acct',
4224                   'dup',
4225                   'dup3',
4226                   'epoll_create1',
4227                   'epoll_ctl',
4228                   'eventfd2',
4229                   'flock',
4230                   'memfd_create',
4231                   'mmap_pgoff',
4232                   'mq_getsetattr',
4233                   'mq_open',
4234                   'msync',
4235                   'open',
4236                   'open_by_handle_at',
4237                   'openat',
4238                   'perf_event_open',
4239                   'pipe2',
4240                   'remap_file_pages',
4241                   'setns',
4242                   'shmat',
4243                   'shmctl',
4244                   'shmdt',
4245                   'socket',
4246                   'socketpair',
4247                   'swapoff',
4248                   'swapon',
4249                   'uselib']),
4250  'recvmmsg': set(['clock_nanosleep',
4251                   'epoll_wait',
4252                   'faccessat',
4253                   'fadvise64_64',
4254                   'fchmod',
4255                   'fchmodat',
4256                   'fchown',
4257                   'fchownat',
4258                   'ftruncate',
4259                   'futex',
4260                   'futimesat',
4261                   'inotify_add_watch',
4262                   'io_getevents',
4263                   'ioctl',
4264                   'linkat',
4265                   'memfd_create',
4266                   'mq_getsetattr',
4267                   'mq_notify',
4268                   'mq_timedreceive',
4269                   'mq_timedsend',
4270                   'mq_unlink',
4271                   'nanosleep',
4272                   'poll',
4273                   'ppoll',
4274                   'pselect6',
4275                   'readlinkat',
4276                   'recvfrom',
4277                   'rt_sigtimedwait',
4278                   'sched_rr_get_interval',
4279                   'select',
4280                   'semtimedop',
4281                   'sendfile',
4282                   'sendfile64',
4283                   'sendto',
```



```
4284                   'settimeofday',
4285                   'stime',
4286                   'swapoff',
4287                   'swapon',
4288                   'timer_gettime',
4289                   'timer_settime',
4290                   'timerfd_gettime',
4291                   'timerfd_settime',
4292                   'unlink',
4293                   'unlinkat',
4294                   'uselib',
4295                   'utime']),
4296  'remap_file_pages': set(['brk',
4297                           'get_mempolicy',
4298                           'madvise',
4299                           'mincore',
4300                           'mlockall',
4301                           'mprotect',
4302                           'mremap',
4303                           'msync',
4304                           'munlock',
4305                           'munlockall',
4306                           'pkey_mprotect',
4307                           'prctl',
4308                           'shmdt']),
4309  'renameat2': set(['acct',
4310                    'mq_open',
4311                    'mq_unlink',
4312                    'open',
4313                    'openat',
4314                    'quotactl',
4315                    'rmdir',
4316                    'swapoff',
4317                    'swapon',
4318                    'symlinkat',
4319                    'sysfs',
4320                    'unlink',
4321                    'unlinkat',
4322                    'uselib']),
4323  'rt_sigqueueinfo': set(['kill',
4324                          'rt_sigreturn',
4325                          'rt_sigtimedwait',
4326                          'rt_tgsigqueueinfo',
4327                          'tgkill',
4328                          'timer_create',
4329                          'tkill']),
4330  'rt_sigreturn': set(['capget',
4331                       'clone',
4332                       'fork',
4333                       'get_robust_list',
4334                       'getitimer',
4335                       'getpgid',
4336                       'getppid',
4337                       'getpriority',
4338                       'getrusage',
4339                       'getsid',
4340                       'ioperm',
4341                       'iopl',
4342                       'ioprio_get',
4343                       'ioprio_set',
4344                       'keyctl',
4345                       'kill',
4346                       'migrate_pages',
4347                       'move_pages',
4348                       'mq_timedreceive',
4349                       'mq_timedsend',
```

```
                      'msgrcv',
                      'perf_event_open',
                      'prctl',
                      'prlimit64',
                      'ptrace',
                      'rt_sigaction',
                      'rt_sigprocmask',
                      'rt_sigtimedwait',
                      'sched_getaffinity',
                      'sched_getattr',
                      'sched_getparam',
                      'sched_getscheduler',
                      'sched_rr_get_interval',
                      'sched_setaffinity',
                      'sched_setattr',
                      'sched_setparam',
                      'sched_setscheduler',
                      'semtimedop',
                      'setitimer',
                      'setns',
                      'setpgid',
                      'setpriority',
                      'setsid',
                      'sigaction',
                      'sigaltstack',
                      'signal',
                      'umount',
                      'vfork']),
 'rt_sigtimedwait': set(['capget',
                         'clone',
                         'fork',
                         'get_robust_list',
                         'getitimer',
                         'getpgid',
                         'getppid',
                         'getpriority',
                         'getrusage',
                         'getsid',
                         'ioperm',
                         'iopl',
                         'ioprio_get',
                         'ioprio_set',
                         'keyctl',
                         'kill',
                         'migrate_pages',
                         'move_pages',
                         'mq_timedreceive',
                         'mq_timedsend',
                         'msgrcv',
                         'perf_event_open',
                         'prctl',
                         'prlimit64',
                         'ptrace',
                         'rt_sigaction',
                         'rt_sigprocmask',
                         'rt_sigqueueinfo',
                         'rt_sigreturn',
                         'rt_tgsigqueueinfo',
                         'sched_getaffinity',
                         'sched_getattr',
                         'sched_getparam',
                         'sched_getscheduler',
                         'sched_rr_get_interval',
                         'sched_setaffinity',
                         'sched_setattr',
                         'sched_setparam',
```



```
                         'sched_setscheduler',
                         'semtimedop',
                         'setitimer',
                         'setns',
                         'setpgid',
                         'setpriority',
                         'setsid',
                         'sigaction',
                         'sigaltstack',
                         'signal',
                         'tgkill',
                         'timer_create',
                         'tkill',
                         'umount',
                         'vfork']),
 'rt_tgsigqueueinfo': set(['kill',
                           'rt_sigqueueinfo',
                           'rt_sigreturn',
                           'rt_sigtimedwait',
                           'tgkill',
                           'timer_create',
                           'tkill']),
 'sched_getattr': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioperm',
                       'iopl',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'migrate_pages',
                       'move_pages',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'msgrcv',
                       'perf_event_open',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'rt_sigaction',
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setattr',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'sigaction',
                       'sigaltstack',
```

```
                       'signal',
                       'umount',
                       'vfork']),
 'sched_getparam': set(['capget',
                        'clone',
                        'fork',
                        'get_robust_list',
                        'getitimer',
                        'getpgid',
                        'getppid',
                        'getpriority',
                        'getrusage',
                        'getsid',
                        'ioprio_get',
                        'ioprio_set',
                        'keyctl',
                        'kill',
                        'migrate_pages',
                        'move_pages',
                        'mq_timedreceive',
                        'mq_timedsend',
                        'msgrcv',
                        'perf_event_open',
                        'prctl',
                        'prlimit64',
                        'ptrace',
                        'rt_sigaction',
                        'rt_sigprocmask',
                        'rt_sigtimedwait',
                        'sched_getaffinity',
                        'sched_getattr',
                        'sched_getscheduler',
                        'sched_rr_get_interval',
                        'sched_setaffinity',
                        'sched_setattr',
                        'sched_setparam',
                        'sched_setscheduler',
                        'semtimedop',
                        'setitimer',
                        'setns',
                        'setpgid',
                        'setpriority',
                        'setsid',
                        'sigaction',
                        'sigaltstack',
                        'signal',
                        'umount',
                        'vfork']),
 'sched_getscheduler': set(['capget',
                            'clone',
                            'fork',
                            'get_robust_list',
                            'getitimer',
                            'getpgid',
                            'getppid',
                            'getpriority',
                            'getrusage',
                            'getsid',
                            'ioprio_get',
                            'ioprio_set',
                            'keyctl',
                            'kill',
                            'migrate_pages',
                            'move_pages',
                            'mq_timedreceive',
                            'mq_timedsend',
```



```
                            'msgrcv',
                            'perf_event_open',
                            'prctl',
                            'prlimit64',
                            'ptrace',
                            'rt_sigaction',
                            'rt_sigprocmask',
                            'rt_sigtimedwait',
                            'sched_getaffinity',
                            'sched_getattr',
                            'sched_getparam',
                            'sched_rr_get_interval',
                            'sched_setaffinity',
                            'sched_setattr',
                            'sched_setparam',
                            'sched_setscheduler',
                            'semtimedop',
                            'setitimer',
                            'setns',
                            'setpgid',
                            'setpriority',
                            'setsid',
                            'sigaction',
                            'sigaltstack',
                            'signal',
                            'umount',
                            'vfork']),
 'sched_setaffinity': set(['capget',
                           'clone',
                           'fork',
                           'get_robust_list',
                           'getitimer',
                           'getpgid',
                           'getppid',
                           'getpriority',
                           'getrusage',
                           'getsid',
                           'ioprio_get',
                           'ioprio_set',
                           'keyctl',
                           'kill',
                           'migrate_pages',
                           'move_pages',
                           'mq_timedreceive',
                           'mq_timedsend',
                           'msgrcv',
                           'perf_event_open',
                           'prctl',
                           'prlimit64',
                           'ptrace',
                           'rt_sigaction',
                           'rt_sigprocmask',
                           'rt_sigtimedwait',
                           'sched_getaffinity',
                           'sched_getattr',
                           'sched_getparam',
                           'sched_getscheduler',
                           'sched_rr_get_interval',
                           'sched_setattr',
                           'sched_setparam',
                           'sched_setscheduler',
                           'semtimedop',
                           'setitimer',
                           'setns',
                           'setpgid',
                           'setpriority',
```

```
                          'setresuid',
                          'setreuid',
                          'setsid',
                          'setuid',
                          'sigaction',
                          'sigaltstack',
                          'signal',
                          'umount',
                          'vfork']),
 'sched_setattr': set(['capget',
                       'clone',
                       'fork',
                       'get_robust_list',
                       'getitimer',
                       'getpgid',
                       'getppid',
                       'getpriority',
                       'getrusage',
                       'getsid',
                       'ioperm',
                       'iopl',
                       'ioprio_get',
                       'ioprio_set',
                       'keyctl',
                       'kill',
                       'migrate_pages',
                       'move_pages',
                       'mq_timedreceive',
                       'mq_timedsend',
                       'msgrcv',
                       'perf_event_open',
                       'prctl',
                       'prlimit64',
                       'ptrace',
                       'rt_sigaction',
                       'rt_sigprocmask',
                       'rt_sigtimedwait',
                       'sched_getaffinity',
                       'sched_getattr',
                       'sched_getparam',
                       'sched_getscheduler',
                       'sched_rr_get_interval',
                       'sched_setaffinity',
                       'sched_setparam',
                       'sched_setscheduler',
                       'semtimedop',
                       'setitimer',
                       'setns',
                       'setpgid',
                       'setpriority',
                       'setsid',
                       'sigaction',
                       'sigaltstack',
                       'signal',
                       'umount',
                       'vfork']),
 'select': set(['capget',
                'clock_nanosleep',
                'clone',
                'epoll_wait',
                'faccessat',
                'fadvise64_64',
                'fchmod',
                'fchmodat',
                'fchown',
                'fchownat',
```



```
                'fork',
                'ftruncate',
                'futex',
                'futimesat',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'inotify_add_watch',
                'io_getevents',
                'ioctl',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'linkat',
                'memfd_create',
                'migrate_pages',
                'move_pages',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'msgrcv',
                'nanosleep',
                'perf_event_open',
                'personality',
                'poll',
                'ppoll',
                'prctl',
                'prlimit64',
                'pselect6',
                'ptrace',
                'readlinkat',
                'recvmmsg',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'sendfile',
                'sendfile64',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'settimeofday',
                'sigaction',
                'sigaltstack',
                'signal',
                'stime',
                'swapoff',
                'swapon',
```

```
4746                 'timer_gettime',
4747                 'timer_settime',
4748                 'timerfd_gettime',
4749                 'timerfd_settime',
4750                 'umount',
4751                 'unlink',
4752                 'unlinkat',
4753                 'uselib',
4754                 'utime',
4755                 'vfork']),
4756  'semctl': set(['semtimedop']),
4757  'semtimedop': set(['clock_nanosleep',
4758                     'epoll_wait',
4759                     'faccessat',
4760                     'fadvise64_64',
4761                     'fchmod',
4762                     'fchmodat',
4763                     'fchown',
4764                     'fchownat',
4765                     'ftruncate',
4766                     'futex',
4767                     'futimesat',
4768                     'inotify_add_watch',
4769                     'io_getevents',
4770                     'ioctl',
4771                     'linkat',
4772                     'memfd_create',
4773                     'mq_getsetattr',
4774                     'mq_notify',
4775                     'mq_timedreceive',
4776                     'mq_timedsend',
4777                     'mq_unlink',
4778                     'msgctl',
4779                     'msgrcv',
4780                     'msgsnd',
4781                     'nanosleep',
4782                     'poll',
4783                     'ppoll',
4784                     'pselect6',
4785                     'readlinkat',
4786                     'recvmmsg',
4787                     'rt_sigtimedwait',
4788                     'sched_rr_get_interval',
4789                     'select',
4790                     'semctl',
4791                     'sendfile',
4792                     'sendfile64',
4793                     'settimeofday',
4794                     'shmat',
4795                     'shmctl',
4796                     'stime',
4797                     'swapoff',
4798                     'swapon',
4799                     'timer_gettime',
4800                     'timer_settime',
4801                     'timerfd_gettime',
4802                     'timerfd_settime',
4803                     'unlink',
4804                     'unlinkat',
4805                     'uselib',
4806                     'utime']),
4807  'sendfile': set(['accept4',
4808                   'acct',
4809                   'dup',
4810                   'dup3',
4811                   'epoll_create1',
```



```
4812                   'epoll_ctl',
4813                   'eventfd2',
4814                   'fadvise64_64',
4815                   'flock',
4816                   'memfd_create',
4817                   'mmap_pgoff',
4818                   'mq_open',
4819                   'msync',
4820                   'open',
4821                   'open_by_handle_at',
4822                   'openat',
4823                   'perf_event_open',
4824                   'pipe2',
4825                   'remap_file_pages',
4826                   'setns',
4827                   'shmat',
4828                   'shmctl',
4829                   'shmdt',
4830                   'socket',
4831                   'socketpair',
4832                   'swapoff',
4833                   'swapon',
4834                   'uselib']),
4835  'sendfile64': set(['accept4',
4836                     'acct',
4837                     'dup',
4838                     'dup3',
4839                     'epoll_create1',
4840                     'epoll_ctl',
4841                     'eventfd2',
4842                     'fadvise64_64',
4843                     'flock',
4844                     'memfd_create',
4845                     'mmap_pgoff',
4846                     'mq_open',
4847                     'msync',
4848                     'open',
4849                     'open_by_handle_at',
4850                     'openat',
4851                     'perf_event_open',
4852                     'pipe2',
4853                     'remap_file_pages',
4854                     'setns',
4855                     'shmat',
4856                     'shmctl',
4857                     'shmdt',
4858                     'socket',
4859                     'socketpair',
4860                     'swapoff',
4861                     'swapon',
4862                     'uselib']),
4863  'sendto': set(['accept4',
4864                 'acct',
4865                 'dup',
4866                 'dup3',
4867                 'epoll_create1',
4868                 'epoll_ctl',
4869                 'eventfd2',
4870                 'flock',
4871                 'memfd_create',
4872                 'mmap_pgoff',
4873                 'mq_getsetattr',
4874                 'mq_open',
4875                 'msync',
4876                 'open',
4877                 'open_by_handle_at',
```

```
                'openat',
                'perf_event_open',
                'pipe2',
                'remap_file_pages',
                'setns',
                'shmat',
                'shmctl',
                'shmdt',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'uselib']),
 'set_mempolicy': set(['get_mempolicy', 'mbind']),
 'set_thread_area': set(['arch_prctl',
                         'capget',
                         'clone',
                         'fork',
                         'get_robust_list',
                         'getitimer',
                         'getpgid',
                         'getppid',
                         'getpriority',
                         'getrusage',
                         'getsid',
                         'ioperm',
                         'iopl',
                         'ioprio_get',
                         'ioprio_set',
                         'keyctl',
                         'kill',
                         'migrate_pages',
                         'move_pages',
                         'mq_timedreceive',
                         'mq_timedsend',
                         'msgrcv',
                         'perf_event_open',
                         'prctl',
                         'prlimit64',
                         'ptrace',
                         'rt_sigaction',
                         'rt_sigprocmask',
                         'rt_sigtimedwait',
                         'sched_getaffinity',
                         'sched_getattr',
                         'sched_getparam',
                         'sched_getscheduler',
                         'sched_rr_get_interval',
                         'sched_setaffinity',
                         'sched_setattr',
                         'sched_setparam',
                         'sched_setscheduler',
                         'semtimedop',
                         'setitimer',
                         'setns',
                         'setpgid',
                         'setpriority',
                         'setsid',
                         'sigaction',
                         'sigaltstack',
                         'signal',
                         'umount',
                         'vfork']),
 'set_trip_temp': set(['get_curr_temp', 'get_trip_temp']),
 'setgroups16': set(['setgroups']),
 'setitimer': set(['adjtimex',
```



```
                   'alarm',
                   'clock_adjtime',
                   'exit_group',
                   'getitimer',
                   'getrusage',
                   'ppoll',
                   'select',
                   'settimeofday',
                   'timer_create',
                   'wait4',
                   'waitid']),
 'setpgid': set(['capget',
                 'clone',
                 'exit_group',
                 'fork',
                 'get_robust_list',
                 'getitimer',
                 'getpgid',
                 'getppid',
                 'getpriority',
                 'getrusage',
                 'getsid',
                 'ioprio_get',
                 'ioprio_set',
                 'keyctl',
                 'kill',
                 'migrate_pages',
                 'move_pages',
                 'mq_timedreceive',
                 'mq_timedsend',
                 'msgrcv',
                 'perf_event_open',
                 'prctl',
                 'prlimit64',
                 'ptrace',
                 'rt_sigaction',
                 'rt_sigprocmask',
                 'rt_sigtimedwait',
                 'sched_getaffinity',
                 'sched_getattr',
                 'sched_getparam',
                 'sched_getscheduler',
                 'sched_rr_get_interval',
                 'sched_setaffinity',
                 'sched_setattr',
                 'sched_setparam',
                 'sched_setscheduler',
                 'semtimedop',
                 'setitimer',
                 'setns',
                 'setpriority',
                 'setresuid',
                 'setreuid',
                 'setsid',
                 'setuid',
                 'sigaction',
                 'sigaltstack',
                 'signal',
                 'timer_create',
                 'umount',
                 'vfork']),
 'setrlimit': set(['getrlimit', 'old_getrlimit', 'prlimit64']),
 'setsid': set(['exit_group', 'timer_create']),
 'setsockopt': set(['accept4']),
 'settimeofday': set(['adjtimex',
                      'alarm',
```

```
                    'clock_adjtime',
                    'getitimer',
                    'getrusage',
                    'ppoll',
                    'select',
                    'setitimer',
                    'wait4',
                    'waitid']),
 'shmctl': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioperm',
                'iopl',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgctl',
                'msgrcv',
                'msgsnd',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semctl',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'shmat',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'shmdt': set(['brk',
               'get_mempolicy',
               'madvise',
               'mincore',
               'mlockall',
               'mprotect',
```



```
               'mremap',
               'msync',
               'munlock',
               'munlockall',
               'pkey_mprotect',
               'prctl',
               'remap_file_pages']),
 'sigaction': set(['capget',
                   'clone',
                   'fork',
                   'get_robust_list',
                   'getitimer',
                   'getpgid',
                   'getppid',
                   'getpriority',
                   'getrusage',
                   'getsid',
                   'ioperm',
                   'iopl',
                   'ioprio_get',
                   'ioprio_set',
                   'keyctl',
                   'kill',
                   'migrate_pages',
                   'move_pages',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msgrcv',
                   'perf_event_open',
                   'prctl',
                   'prlimit64',
                   'ptrace',
                   'rt_sigaction',
                   'rt_sigprocmask',
                   'rt_sigtimedwait',
                   'sched_getaffinity',
                   'sched_getattr',
                   'sched_getparam',
                   'sched_getscheduler',
                   'sched_rr_get_interval',
                   'sched_setaffinity',
                   'sched_setattr',
                   'sched_setparam',
                   'sched_setscheduler',
                   'semtimedop',
                   'setitimer',
                   'setns',
                   'setpgid',
                   'setpriority',
                   'setsid',
                   'sigaltstack',
                   'signal',
                   'umount',
                   'vfork']),
 'signalfd4': set(['accept4',
                   'bind',
                   'bpf',
                   'connect',
                   'copy_file_range',
                   'epoll_ctl',
                   'epoll_wait',
                   'fadvise64_64',
                   'fallocate',
                   'fchdir',
                   'fchmod',
                   'fchown',
```

```
                    'fcntl',
                    'fcntl64',
                    'fdatasync',
                    'fgetxattr',
                    'flistxattr',
                    'flock',
                    'fremovexattr',
                    'fsetxattr',
                    'fstatfs',
                    'fstatfs64',
                    'fsync',
                    'ftruncate',
                    'futimesat',
                    'getdents',
                    'getdents64',
                    'getpeername',
                    'getsockname',
                    'getsockopt',
                    'inotify_add_watch',
                    'inotify_rm_watch',
                    'ioctl',
                    'listen',
                    'llseek',
                    'lseek',
                    'mq_getsetattr',
                    'mq_notify',
                    'mq_timedreceive',
                    'mq_timedsend',
                    'old_readdir',
                    'perf_event_open',
                    'pread64',
                    'preadv',
                    'preadv2',
                    'preadv64',
                    'preadv64v2',
                    'pwrite64',
                    'pwritev',
                    'pwritev2',
                    'pwritev64',
                    'pwritev64v2',
                    'read',
                    'readahead',
                    'readv',
                    'recvfrom',
                    'sendfile',
                    'sendfile64',
                    'sendto',
                    'setsockopt',
                    'shutdown',
                    'splice',
                    'sync_file_range',
                    'syncfs',
                    'tee',
                    'utime',
                    'utimensat',
                    'vmsplice',
                    'write',
                    'writev']),
 'splice': set(['accept4',
                'acct',
                'bind',
                'bpf',
                'connect',
                'copy_file_range',
                'dup',
                'dup3',
```



```
                'epoll_create1',
                'epoll_ctl',
                'epoll_wait',
                'eventfd2',
                'fadvise64_64',
                'fallocate',
                'fchdir',
                'fchmod',
                'fchown',
                'fcntl',
                'fcntl64',
                'fdatasync',
                'fgetxattr',
                'flistxattr',
                'flock',
                'fremovexattr',
                'fsetxattr',
                'fstatfs',
                'fstatfs64',
                'fsync',
                'ftruncate',
                'futimesat',
                'getdents',
                'getdents64',
                'getpeername',
                'getsockname',
                'getsockopt',
                'inotify_add_watch',
                'inotify_rm_watch',
                'ioctl',
                'listen',
                'llseek',
                'lseek',
                'memfd_create',
                'mmap_pgoff',
                'mq_getsetattr',
                'mq_notify',
                'mq_open',
                'mq_timedreceive',
                'mq_timedsend',
                'msync',
                'old_readdir',
                'open',
                'open_by_handle_at',
                'openat',
                'perf_event_open',
                'pipe2',
                'pread64',
                'preadv',
                'preadv2',
                'preadv64',
                'preadv64v2',
                'pwrite64',
                'pwritev',
                'pwritev2',
                'pwritev64',
                'pwritev64v2',
                'read',
                'readahead',
                'readv',
                'recvfrom',
                'remap_file_pages',
                'sendfile',
                'sendfile64',
                'sendto',
                'setns',
```

```
                'setsockopt',
                'shmat',
                'shmctl',
                'shmdt',
                'shutdown',
                'signalfd4',
                'socket',
                'socketpair',
                'swapoff',
                'swapon',
                'sync_file_range',
                'syncfs',
                'tee',
                'uselib',
                'utime',
                'utimensat',
                'vmsplice',
                'write',
                'writev']),
 'stat': set(['fstat', 'lstat']),
 'statfs': set(['fstatfs', 'fstatfs64', 'statfs64', 'ustat']),
 'statfs64': set(['fstatfs', 'fstatfs64', 'statfs', 'ustat']),
 'swapoff': set(['swapon']),
 'swapon': set(['faccessat',
                'fadvise64_64',
                'fchmod',
                'fchmodat',
                'fchown',
                'fchownat',
                'ftruncate',
                'inotify_add_watch',
                'ioctl',
                'linkat',
                'memfd_create',
                'mq_getsetattr',
                'mq_notify',
                'mq_timedreceive',
                'mq_timedsend',
                'mq_unlink',
                'readlinkat',
                'sendfile',
                'sendfile64',
                'swapoff',
                'unlink',
                'unlinkat',
                'uselib']),
 'symlinkat': set(['acct',
                   'mq_open',
                   'mq_unlink',
                   'open',
                   'openat',
                   'quotactl',
                   'renameat2',
                   'rmdir',
                   'swapoff',
                   'swapon',
                   'sysfs',
                   'unlink',
                   'unlinkat',
                   'uselib']),
 'sync_file_range': set(['accept4',
                         'bind',
                         'bpf',
                         'connect',
                         'copy_file_range',
                         'epoll_ctl',
```



```
                         'epoll_wait',
                         'fadvise64_64',
                         'fallocate',
                         'fchdir',
                         'fchmod',
                         'fchown',
                         'fcntl',
                         'fcntl64',
                         'fdatasync',
                         'fgetxattr',
                         'flistxattr',
                         'flock',
                         'fremovexattr',
                         'fsetxattr',
                         'fstatfs',
                         'fstatfs64',
                         'fsync',
                         'ftruncate',
                         'futimesat',
                         'getdents',
                         'getdents64',
                         'getpeername',
                         'getsockname',
                         'getsockopt',
                         'inotify_add_watch',
                         'inotify_rm_watch',
                         'ioctl',
                         'listen',
                         'llseek',
                         'lseek',
                         'mq_getsetattr',
                         'mq_notify',
                         'mq_timedreceive',
                         'mq_timedsend',
                         'old_readdir',
                         'perf_event_open',
                         'pread64',
                         'preadv',
                         'preadv2',
                         'preadv64',
                         'preadv64v2',
                         'pwrite64',
                         'pwritev',
                         'pwritev2',
                         'pwritev64',
                         'pwritev64v2',
                         'read',
                         'readahead',
                         'readv',
                         'recvfrom',
                         'sendfile',
                         'sendfile64',
                         'sendto',
                         'setsockopt',
                         'shutdown',
                         'signalfd4',
                         'splice',
                         'syncfs',
                         'tee',
                         'utime',
                         'utimensat',
                         'vmsplice',
                         'write',
                         'writev']),
 'syncfs': set(['accept4',
                'bind',
```

```
5406                 'bpf',
5407                 'connect',
5408                 'copy_file_range',
5409                 'epoll_ctl',
5410                 'epoll_wait',
5411                 'fadvise64_64',
5412                 'fallocate',
5413                 'fchdir',
5414                 'fchmod',
5415                 'fchown',
5416                 'fcntl',
5417                 'fcntl64',
5418                 'fdatasync',
5419                 'fgetxattr',
5420                 'flistxattr',
5421                 'flock',
5422                 'fremovexattr',
5423                 'fsetxattr',
5424                 'fstatfs',
5425                 'fstatfs64',
5426                 'fsync',
5427                 'ftruncate',
5428                 'futimesat',
5429                 'getdents',
5430                 'getdents64',
5431                 'getpeername',
5432                 'getsockname',
5433                 'getsockopt',
5434                 'inotify_add_watch',
5435                 'inotify_rm_watch',
5436                 'ioctl',
5437                 'listen',
5438                 'llseek',
5439                 'lseek',
5440                 'mq_getsetattr',
5441                 'mq_notify',
5442                 'mq_timedreceive',
5443                 'mq_timedsend',
5444                 'old_readdir',
5445                 'perf_event_open',
5446                 'pread64',
5447                 'preadv',
5448                 'preadv2',
5449                 'preadv64',
5450                 'preadv64v2',
5451                 'pwrite64',
5452                 'pwritev',
5453                 'pwritev2',
5454                 'pwritev64',
5455                 'pwritev64v2',
5456                 'read',
5457                 'readahead',
5458                 'readv',
5459                 'recvfrom',
5460                 'sendfile',
5461                 'sendfile64',
5462                 'sendto',
5463                 'setsockopt',
5464                 'shutdown',
5465                 'signalfd4',
5466                 'splice',
5467                 'sync_file_range',
5468                 'tee',
5469                 'utime',
5470                 'utimensat',
5471                 'vmsplice',
```



```
5472                  'write',
5473                  'writev']),
5474  'sysfs': set(['acct',
5475                'mq_open',
5476                'mq_unlink',
5477                'open',
5478                'openat',
5479                'quotactl',
5480                'renameat2',
5481                'rmdir',
5482                'swapoff',
5483                'swapon',
5484                'symlinkat',
5485                'unlink',
5486                'unlinkat',
5487                'uselib']),
5488  'sysinfo': set(['capget',
5489                  'clock_nanosleep',
5490                  'clone',
5491                  'epoll_wait',
5492                  'faccessat',
5493                  'fadvise64_64',
5494                  'fchmod',
5495                  'fchmodat',
5496                  'fchown',
5497                  'fchownat',
5498                  'fork',
5499                  'ftruncate',
5500                  'futex',
5501                  'futimesat',
5502                  'get_robust_list',
5503                  'getitimer',
5504                  'getpgid',
5505                  'getppid',
5506                  'getpriority',
5507                  'getrusage',
5508                  'getsid',
5509                  'inotify_add_watch',
5510                  'io_getevents',
5511                  'ioctl',
5512                  'ioperm',
5513                  'iopl',
5514                  'ioprio_get',
5515                  'ioprio_set',
5516                  'keyctl',
5517                  'kill',
5518                  'linkat',
5519                  'memfd_create',
5520                  'migrate_pages',
5521                  'move_pages',
5522                  'mq_getsetattr',
5523                  'mq_notify',
5524                  'mq_timedreceive',
5525                  'mq_timedsend',
5526                  'mq_unlink',
5527                  'msgrcv',
5528                  'nanosleep',
5529                  'perf_event_open',
5530                  'poll',
5531                  'ppoll',
5532                  'prctl',
5533                  'prlimit64',
5534                  'pselect6',
5535                  'ptrace',
5536                  'readlinkat',
5537                  'recvmmsg',
```

```
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'select',
                'semtimedop',
                'sendfile',
                'sendfile64',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'settimeofday',
                'sigaction',
                'sigaltstack',
                'signal',
                'stime',
                'swapoff',
                'swapon',
                'timer_gettime',
                'timer_settime',
                'timerfd_gettime',
                'timerfd_settime',
                'umount',
                'unlink',
                'unlinkat',
                'uselib',
                'utime',
                'vfork']),
 'syslog': set(['capget',
                'clone',
                'fork',
                'get_robust_list',
                'getitimer',
                'getpgid',
                'getppid',
                'getpriority',
                'getrusage',
                'getsid',
                'ioperm',
                'iopl',
                'ioprio_get',
                'ioprio_set',
                'keyctl',
                'kill',
                'migrate_pages',
                'move_pages',
                'mq_timedreceive',
                'mq_timedsend',
                'msgrcv',
                'perf_event_open',
                'prctl',
                'prlimit64',
                'ptrace',
                'rt_sigaction',
                'rt_sigprocmask',
                'rt_sigtimedwait',
```



```
                'sched_getaffinity',
                'sched_getattr',
                'sched_getparam',
                'sched_getscheduler',
                'sched_rr_get_interval',
                'sched_setaffinity',
                'sched_setattr',
                'sched_setparam',
                'sched_setscheduler',
                'semtimedop',
                'setitimer',
                'setns',
                'setpgid',
                'setpriority',
                'setsid',
                'sigaction',
                'sigaltstack',
                'signal',
                'umount',
                'vfork']),
 'tee': set(['accept4',
             'acct',
             'bind',
             'bpf',
             'connect',
             'copy_file_range',
             'dup',
             'dup3',
             'epoll_create1',
             'epoll_ctl',
             'epoll_wait',
             'eventfd2',
             'fadvise64_64',
             'fallocate',
             'fchdir',
             'fchmod',
             'fchown',
             'fcntl',
             'fcntl64',
             'fdatasync',
             'fgetxattr',
             'flistxattr',
             'flock',
             'fremovexattr',
             'fsetxattr',
             'fstatfs',
             'fstatfs64',
             'fsync',
             'ftruncate',
             'futimesat',
             'getdents',
             'getdents64',
             'getpeername',
             'getsockname',
             'getsockopt',
             'inotify_add_watch',
             'inotify_rm_watch',
             'ioctl',
             'listen',
             'llseek',
             'lseek',
             'memfd_create',
             'mmap_pgoff',
             'mq_getsetattr',
             'mq_notify',
             'mq_open',
```

```
5670              'mq_timedreceive',
5671              'mq_timedsend',
5672              'msync',
5673              'old_readdir',
5674              'open',
5675              'open_by_handle_at',
5676              'openat',
5677              'perf_event_open',
5678              'pipe2',
5679              'pread64',
5680              'preadv',
5681              'preadv2',
5682              'preadv64',
5683              'preadv64v2',
5684              'pwrite64',
5685              'pwritev',
5686              'pwritev2',
5687              'pwritev64',
5688              'pwritev64v2',
5689              'read',
5690              'readahead',
5691              'readv',
5692              'recvfrom',
5693              'remap_file_pages',
5694              'sendfile',
5695              'sendfile64',
5696              'sendto',
5697              'setns',
5698              'setsockopt',
5699              'shmat',
5700              'shmctl',
5701              'shmdt',
5702              'shutdown',
5703              'signalfd4',
5704              'socket',
5705              'socketpair',
5706              'splice',
5707              'swapoff',
5708              'swapon',
5709              'sync_file_range',
5710              'syncfs',
5711              'uselib',
5712              'utime',
5713              'utimensat',
5714              'vmsplice',
5715              'write',
5716              'writev']),
5717  'timer_create': set(['clock_adjtime',
5718                       'clock_getres',
5719                       'clock_gettime',
5720                       'clock_nanosleep',
5721                       'clock_settime',
5722                       'timer_delete',
5723                       'timer_gettime',
5724                       'timer_settime']),
5725  'timer_delete': set(['clock_adjtime',
5726                       'clock_getres',
5727                       'clock_gettime',
5728                       'clock_nanosleep',
5729                       'clock_settime',
5730                       'timer_create',
5731                       'timer_gettime',
5732                       'timer_settime']),
5733  'timer_gettime': set(['clock_adjtime',
5734                        'clock_getres',
5735                        'clock_gettime',
```



```
5736                        'clock_nanosleep',
5737                        'clock_settime',
5738                        'timer_create',
5739                        'timer_delete',
5740                        'timer_settime']),
5741  'timer_settime': set(['clock_adjtime',
5742                        'clock_getres',
5743                        'clock_gettime',
5744                        'clock_nanosleep',
5745                        'clock_settime',
5746                        'timer_create',
5747                        'timer_delete',
5748                        'timer_gettime']),
5749  'timerfd_create': set(['timerfd_gettime', 'timerfd_settime']),
5750  'timerfd_gettime': set(['timerfd_create', 'timerfd_settime']),
5751  'timerfd_settime': set(['timerfd_create', 'timerfd_gettime']),
5752  'umount': set(['acct',
5753                 'capget',
5754                 'clone',
5755                 'fadvise64_64',
5756                 'fork',
5757                 'get_robust_list',
5758                 'getcwd',
5759                 'getitimer',
5760                 'getpgid',
5761                 'getppid',
5762                 'getpriority',
5763                 'getrusage',
5764                 'getsid',
5765                 'ioprio_get',
5766                 'ioprio_set',
5767                 'keyctl',
5768                 'kill',
5769                 'migrate_pages',
5770                 'move_pages',
5771                 'mq_open',
5772                 'mq_timedreceive',
5773                 'mq_timedsend',
5774                 'mq_unlink',
5775                 'msgrcv',
5776                 'perf_event_open',
5777                 'pivot_root',
5778                 'prctl',
5779                 'prlimit64',
5780                 'ptrace',
5781                 'quotactl',
5782                 'rt_sigaction',
5783                 'rt_sigprocmask',
5784                 'rt_sigtimedwait',
5785                 'sched_getaffinity',
5786                 'sched_getattr',
5787                 'sched_getparam',
5788                 'sched_getscheduler',
5789                 'sched_rr_get_interval',
5790                 'sched_setaffinity',
5791                 'sched_setattr',
5792                 'sched_setparam',
5793                 'sched_setscheduler',
5794                 'semtimedop',
5795                 'setitimer',
5796                 'setns',
5797                 'setpgid',
5798                 'setpriority',
5799                 'setresuid',
5800                 'setreuid',
5801                 'setsid',
```

```
                'setuid',
                'sigaction',
                'sigaltstack',
                'signal',
                'swapon',
                'syncfs',
                'ustat',
                'vfork']),
 'uname': set(['capget',
               'clone',
               'fork',
               'get_robust_list',
               'getitimer',
               'getpgid',
               'getppid',
               'getpriority',
               'getrusage',
               'getsid',
               'ioprio_get',
               'ioprio_set',
               'keyctl',
               'kill',
               'migrate_pages',
               'move_pages',
               'mq_timedreceive',
               'mq_timedsend',
               'msgrcv',
               'perf_event_open',
               'personality',
               'prctl',
               'prlimit64',
               'ptrace',
               'rt_sigaction',
               'rt_sigprocmask',
               'rt_sigtimedwait',
               'sched_getaffinity',
               'sched_getattr',
               'sched_getparam',
               'sched_getscheduler',
               'sched_rr_get_interval',
               'sched_setaffinity',
               'sched_setattr',
               'sched_setparam',
               'sched_setscheduler',
               'semtimedop',
               'setitimer',
               'setns',
               'setpgid',
               'setpriority',
               'setsid',
               'sigaction',
               'sigaltstack',
               'signal',
               'umount',
               'vfork']),
 'uselib': set(['acct',
                'fadvise64_64',
                'getcwd',
                'mq_open',
                'mq_unlink',
                'pivot_root',
                'quotactl',
                'swapon',
                'syncfs',
                'umount',
                'ustat']),
```



```
 'utime': set(['clock_nanosleep',
               'epoll_wait',
               'faccessat',
               'fadvise64_64',
               'fchmod',
               'fchmodat',
               'fchown',
               'fchownat',
               'ftruncate',
               'futex',
               'futimesat',
               'inotify_add_watch',
               'io_getevents',
               'ioctl',
               'linkat',
               'memfd_create',
               'mq_getsetattr',
               'mq_notify',
               'mq_timedreceive',
               'mq_timedsend',
               'mq_unlink',
               'nanosleep',
               'poll',
               'ppoll',
               'pselect6',
               'readlinkat',
               'recvmmsg',
               'rt_sigtimedwait',
               'sched_rr_get_interval',
               'select',
               'semtimedop',
               'sendfile',
               'sendfile64',
               'settimeofday',
               'stime',
               'swapoff',
               'swapon',
               'timer_gettime',
               'timer_settime',
               'timerfd_gettime',
               'timerfd_settime',
               'unlink',
               'unlinkat',
               'uselib']),
 'utimensat': set(['clock_nanosleep',
                   'epoll_wait',
                   'faccessat',
                   'fadvise64_64',
                   'fchmod',
                   'fchmodat',
                   'fchown',
                   'fchownat',
                   'ftruncate',
                   'futex',
                   'futimesat',
                   'inotify_add_watch',
                   'io_getevents',
                   'ioctl',
                   'linkat',
                   'memfd_create',
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'mq_unlink',
                   'nanosleep',
```

```
                   'poll',
                   'ppoll',
                   'pselect6',
                   'readlinkat',
                   'recvmmsg',
                   'rt_sigtimedwait',
                   'sched_rr_get_interval',
                   'select',
                   'semtimedop',
                   'sendfile',
                   'sendfile64',
                   'settimeofday',
                   'stime',
                   'swapoff',
                   'swapon',
                   'timer_gettime',
                   'timer_settime',
                   'timerfd_gettime',
                   'timerfd_settime',
                   'unlink',
                   'unlinkat',
                   'uselib',
                   'utime']),
 'vmsplice': set(['accept4',
                  'acct',
                  'bind',
                  'bpf',
                  'connect',
                  'copy_file_range',
                  'dup',
                  'dup3',
                  'epoll_create1',
                  'epoll_ctl',
                  'epoll_wait',
                  'eventfd2',
                  'fadvise64_64',
                  'fallocate',
                  'fchdir',
                  'fchmod',
                  'fchown',
                  'fcntl',
                  'fcntl64',
                  'fdatasync',
                  'fgetxattr',
                  'flistxattr',
                  'flock',
                  'fremovexattr',
                  'fsetxattr',
                  'fstatfs',
                  'fstatfs64',
                  'fsync',
                  'ftruncate',
                  'futimesat',
                  'getdents',
                  'getdents64',
                  'getpeername',
                  'getsockname',
                  'getsockopt',
                  'inotify_add_watch',
                  'inotify_rm_watch',
                  'ioctl',
                  'listen',
                  'llseek',
                  'lseek',
                  'memfd_create',
                  'mmap_pgoff',
```



```
                   'mq_getsetattr',
                   'mq_notify',
                   'mq_open',
                   'mq_timedreceive',
                   'mq_timedsend',
                   'msync',
                   'old_readdir',
                   'open',
                   'open_by_handle_at',
                   'openat',
                   'perf_event_open',
                   'pipe2',
                   'pread64',
                   'preadv',
                   'preadv2',
                   'preadv64',
                   'preadv64v2',
                   'pwrite64',
                   'pwritev',
                   'pwritev2',
                   'pwritev64',
                   'pwritev64v2',
                   'read',
                   'readahead',
                   'readv',
                   'recvfrom',
                   'remap_file_pages',
                   'sendfile',
                   'sendfile64',
                   'sendto',
                   'setns',
                   'setsockopt',
                   'shmat',
                   'shmctl',
                   'shmdt',
                   'shutdown',
                   'signalfd4',
                   'socket',
                   'socketpair',
                   'splice',
                   'swapoff',
                   'swapon',
                   'sync_file_range',
                   'syncfs',
                   'tee',
                   'uselib',
                   'utime',
                   'utimensat',
                   'write',
                   'writev']),
 'write': set(['accept4',
               'bind',
               'bpf',
               'connect',
               'copy_file_range',
               'epoll_ctl',
               'epoll_wait',
               'fadvise64_64',
               'fallocate',
               'fchdir',
               'fchmod',
               'fchown',
               'fcntl',
               'fcntl64',
               'fdatasync',
               'fgetxattr',
```

```
6066                'flistxattr',
6067                'flock',
6068                'fremovexattr',
6069                'fsetxattr',
6070                'fstatfs',
6071                'fstatfs64',
6072                'fsync',
6073                'ftruncate',
6074                'futimesat',
6075                'getdents',
6076                'getdents64',
6077                'getpeername',
6078                'getsockname',
6079                'getsockopt',
6080                'inotify_add_watch',
6081                'inotify_rm_watch',
6082                'ioctl',
6083                'listen',
6084                'llseek',
6085                'lseek',
6086                'mq_getsetattr',
6087                'mq_notify',
6088                'mq_timedreceive',
6089                'mq_timedsend',
6090                'old_readdir',
6091                'perf_event_open',
6092                'pread64',
6093                'preadv',
6094                'preadv2',
6095                'preadv64',
6096                'preadv64v2',
6097                'pwrite64',
6098                'pwritev',
6099                'pwritev2',
6100                'pwritev64',
6101                'pwritev64v2',
6102                'read',
6103                'readahead',
6104                'readv',
6105                'recvfrom',
6106                'sendfile',
6107                'sendfile64',
6108                'sendto',
6109                'setsockopt',
6110                'shutdown',
6111                'signalfd4',
6112                'splice',
6113                'sync_file_range',
6114                'syncfs',
6115                'tee',
6116                'utime',
6117                'utimensat',
6118                'vmsplice',
6119                'writev'])}
```

```
**********************************************************
  618788 Fri Dec 21 15:00:34 2018
new/usr/src/tools/smatch/src/smatch_scripts/implicit_dependencies/without_struct
s/implicit_dependencies_verbose.pretty
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
{'acct': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
          {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
          {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
          {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
          {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
          {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
          {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
          {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
          {'call': 'flock', 'reason': set([('file', 'f_mode')])},
          {'call': 'openat', 'reason': set([('file', 'f_mode')])},
          {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
          {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
          {'call': 'socket', 'reason': set([('file', 'f_mode')])},
          {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
          {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
          {'call': 'open', 'reason': set([('file', 'f_mode')])},
          {'call': 'dup', 'reason': set([('file', 'f_mode')])},
          {'call': 'setns', 'reason': set([('file', 'f_mode')])},
          {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
          {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
          {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
          {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
          {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
          {'call': 'msync', 'reason': set([('file', 'f_mode')])},
          {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])}],
 'alarm': [{'call': 'settimeofday',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'adjtimex',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'waitid',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'getitimer',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'select',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'wait4',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'getrusage',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'setitimer',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'clock_adjtime',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])},
           {'call': 'ppoll',
            'reason': set([('timeval', 'tv_sec'), ('timeval', 'tv_usec')])}],
 'bpf': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'rt_sigtimedwait',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'swapoff', 'reason': set([('file', 'private_data')])},
         {'call': 'sched_getaffinity',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_setparam', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'memfd_create', 'reason': set([('file', 'private_data')])},
         {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
```



```
         {'call': 'remap_file_pages',
          'reason': set([('file', 'private_data')])},
         {'call': 'dup3', 'reason': set([('file', 'private_data')])},
         {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'mq_timedreceive',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_setaffinity',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'semtimedop', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'socketpair', 'reason': set([('file', 'private_data')])},
         {'call': 'sched_rr_get_interval',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'epoll_create1', 'reason': set([('file', 'private_data')])},
         {'call': 'epoll_ctl', 'reason': set([('file', 'private_data')])},
         {'call': 'flock', 'reason': set([('file', 'private_data')])},
         {'call': 'openat', 'reason': set([('file', 'private_data')])},
         {'call': 'uselib', 'reason': set([('file', 'private_data')])},
         {'call': 'rt_sigprocmask', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'accept4', 'reason': set([('file', 'private_data')])},
         {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_setattr', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'migrate_pages', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'shmat', 'reason': set([('file', 'private_data')])},
         {'call': 'socket', 'reason': set([('file', 'private_data')])},
         {'call': 'pipe2', 'reason': set([('file', 'private_data')])},
         {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'perf_event_open',
          'reason': set([('file', 'private_data'), ('mm_segment_t', 'seg')])},
         {'call': 'shmdt', 'reason': set([('file', 'private_data')])},
         {'call': 'rt_sigaction', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'acct', 'reason': set([('file', 'private_data')])},
         {'call': 'open', 'reason': set([('file', 'private_data')])},
         {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'dup', 'reason': set([('file', 'private_data')])},
         {'call': 'setns',
          'reason': set([('file', 'private_data'), ('mm_segment_t', 'seg')])},
         {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'get_robust_list',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'mq_timedsend', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_getscheduler',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'shmctl', 'reason': set([('file', 'private_data')])},
         {'call': 'swapon', 'reason': set([('file', 'private_data')])},
         {'call': 'sched_getattr', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'sched_setscheduler',
          'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'eventfd2', 'reason': set([('file', 'private_data')])},
         {'call': 'mmap_pgoff', 'reason': set([('file', 'private_data')])},
         {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
         {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
```

```
        {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'mq_open', 'reason': set([('file', 'private_data')])},
        {'call': 'msync', 'reason': set([('file', 'private_data')])},
        {'call': 'sched_getparam', 'reason': set([('mm_segment_t', 'seg')])},
        {'call': 'open_by_handle_at',
         'reason': set([('file', 'private_data')])}],
 'brk': [{'call': 'swapoff',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'remap_file_pages',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data'),
                         ('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'io_getevents',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'migrate_pages',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'shmdt',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data'),
                         ('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'get_mempolicy',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data'),
                         ('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'munlockall',
          'reason': set([('mm_struct', 'def_flags'),
                         ('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'pkey_mprotect',
          'reason': set([('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'madvise',
          'reason': set([('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'getrusage',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'io_setup',
```



```
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'mprotect',
          'reason': set([('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'mremap',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data'),
                         ('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'io_destroy',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'mbind',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'prctl',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data'),
                         ('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'move_pages',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'modify_ldt',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data')])},
         {'call': 'munlock',
          'reason': set([('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'mincore',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data'),
                         ('vm_area_struct', 'vm_flags'),
                         ('vm_area_struct', 'vm_start')])},
         {'call': 'msync',
          'reason': set([('mm_struct', 'brk'),
                         ('mm_struct', 'def_flags'),
                         ('mm_struct', 'end_data'),
                         ('mm_struct', 'start_brk'),
                         ('mm_struct', 'start_data'),
                         ('vm_area_struct', 'vm_flags'),
```

```
                        ('vm_area_struct', 'vm_start')])},
        {'call': 'io_cancel',
         'reason': set([('mm_struct', 'brk'),
                        ('mm_struct', 'def_flags'),
                        ('mm_struct', 'end_data'),
                        ('mm_struct', 'start_brk'),
                        ('mm_struct', 'start_data')])},
        {'call': 'mlockall',
         'reason': set([('mm_struct', 'def_flags'),
                        ('vm_area_struct', 'vm_flags'),
                        ('vm_area_struct', 'vm_start')])}],
 'clock_adjtime': [{'call': 'clock_getres',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_delete',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_create',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'clock_gettime',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_settime',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'timer_gettime',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'clock_settime',
                    'reason': set([('k_clock', 'clock_adj')])},
                   {'call': 'clock_nanosleep',
                    'reason': set([('k_clock', 'clock_adj')])}],
 'clock_nanosleep': [{'call': 'rt_sigtimedwait',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'fadvise64_64',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'mq_unlink',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'swapoff',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'clock_getres',
                      'reason': set([('k_clock', 'nsleep')])},
                     {'call': 'timer_delete',
                      'reason': set([('k_clock', 'nsleep')])},
                     {'call': 'fchmod',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'memfd_create',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'readlinkat',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'io_getevents',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'fchown',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'mq_timedreceive',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'utime',
                      'reason': set([('timespec', 'tv_nsec'),
                                     ('timespec', 'tv_sec')])},
                     {'call': 'semtimedop',
                      'reason': set([('timespec', 'tv_nsec'),
```



```
                                     ('timespec', 'tv_sec')])},
                    {'call': 'settimeofday',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'timer_create',
                     'reason': set([('k_clock', 'nsleep')])},
                    {'call': 'clock_gettime',
                     'reason': set([('k_clock', 'nsleep')])},
                    {'call': 'sched_rr_get_interval',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'timerfd_gettime',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'pselect6',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'uselib',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'fchmodat',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'inotify_add_watch',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'timer_settime',
                     'reason': set([('k_clock', 'nsleep'),
                                    ('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'ftruncate',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'timer_gettime',
                     'reason': set([('k_clock', 'nsleep'),
                                    ('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'ioctl',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'linkat',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'stime',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'futimesat',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'poll',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'clock_settime',
                     'reason': set([('k_clock', 'nsleep')])},
                    {'call': 'select',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'unlink',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'nanosleep',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'mq_getsetattr',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
```

```
                    {'call': 'faccessat',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'swapon',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'epoll_wait',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'fchownat',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'timerfd_settime',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'mq_notify',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'sendfile',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'unlinkat',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'clock_adjtime',
                     'reason': set([('k_clock', 'nsleep')])},
                    {'call': 'futex',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'recvmmsg',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'sendfile64',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])},
                    {'call': 'ppoll',
                     'reason': set([('timespec', 'tv_nsec'),
                                    ('timespec', 'tv_sec')])}],
 'clock_settime': [{'call': 'clock_getres',
                    'reason': set([('k_clock', 'clock_set')])},
                   {'call': 'timer_delete',
                    'reason': set([('k_clock', 'clock_set')])},
                   {'call': 'timer_create',
                    'reason': set([('k_clock', 'clock_set')])},
                   {'call': 'clock_gettime',
                    'reason': set([('k_clock', 'clock_set')])},
                   {'call': 'timer_settime',
                    'reason': set([('k_clock', 'clock_set')])},
                   {'call': 'timer_gettime',
                    'reason': set([('k_clock', 'clock_set')])},
                   {'call': 'clock_nanosleep',
                    'reason': set([('k_clock', 'clock_set')])},
                   {'call': 'clock_adjtime',
                    'reason': set([('k_clock', 'clock_set')])}],
 'copy_file_range': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                     {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                     {'call': 'fadvise64_64',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                     {'call': 'fremovexattr',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'readahead', 'reason': set([('fd', 'flags')])},
                     {'call': 'getdents', 'reason': set([('fd', 'flags')])},
```



```
                    {'call': 'writev', 'reason': set([('fd', 'flags')])},
                    {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                    {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                    {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                    {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                    {'call': 'read', 'reason': set([('fd', 'flags')])},
                    {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                    {'call': 'mq_timedreceive',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'utime', 'reason': set([('fd', 'flags')])},
                    {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                    {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                    {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                    {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                    {'call': 'tee', 'reason': set([('fd', 'flags')])},
                    {'call': 'sync_file_range',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                    {'call': 'connect', 'reason': set([('fd', 'flags')])},
                    {'call': 'getsockname',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
                    {'call': 'flock', 'reason': set([('fd', 'flags')])},
                    {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                    {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                    {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                    {'call': 'old_readdir',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'inotify_rm_watch',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                    {'call': 'inotify_add_watch',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                    {'call': 'splice', 'reason': set([('fd', 'flags')])},
                    {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                    {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                    {'call': 'getpeername',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                    {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                    {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                    {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                    {'call': 'perf_event_open',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'pwritev64v2',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                    {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                    {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                    {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                    {'call': 'mq_getsetattr',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                    {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                    {'call': 'listen', 'reason': set([('fd', 'flags')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                    {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                    {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                    {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                    {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                    {'call': 'readv', 'reason': set([('fd', 'flags')])},
```

```
                    {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                    {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                    {'call': 'write', 'reason': set([('fd', 'flags')])},
                    {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                    {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                    {'call': 'bind', 'reason': set([('fd', 'flags')])},
                    {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'delete_module': [{'call': 'init_module',
                    'reason': set([('module', 'exit'),
                                   ('module', 'init'),
                                   ('module', 'state')])},
                   {'call': 'finit_module',
                    'reason': set([('module', 'exit'),
                                   ('module', 'init'),
                                   ('module', 'state')])}],
 'dup3': [{'call': 'unshare',
           'reason': set([('fdtable', 'max_fds'),
                          ('files_struct', 'resize_in_progress')])},
          {'call': 'select', 'reason': set([('fdtable', 'max_fds')])},
          {'call': 'dup2',
           'reason': set([('fdtable', 'max_fds'),
                          ('files_struct', 'resize_in_progress')])}],
 'epoll_ctl': [{'call': 'syncfs','reason': set([('fd', 'flags')])},
               {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
               {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
               {'call': 'readahead', 'reason': set([('fd', 'flags')])},
               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
               {'call': 'writev', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
               {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
               {'call': 'pread64', 'reason': set([('fd', 'flags')])},
               {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
               {'call': 'read', 'reason': set([('fd', 'flags')])},
               {'call': 'fchown', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
               {'call': 'utime', 'reason': set([('fd', 'flags')])},
               {'call': 'fsync', 'reason': set([('fd', 'flags')])},
               {'call': 'bpf', 'reason': set([('fd', 'flags')])},
               {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
               {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'sendto', 'reason': set([('fd', 'flags')])},
               {'call': 'epoll_create1',
                'reason': set([('epitem', 'nwait'),
                               ('epoll_event', 'events')])},
               {'call': 'tee', 'reason': set([('fd', 'flags')])},
               {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
               {'call': 'lseek', 'reason': set([('fd', 'flags')])},
               {'call': 'connect', 'reason': set([('fd', 'flags')])},
               {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
               {'call': 'flock', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
               {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
               {'call': 'accept4', 'reason': set([('fd', 'flags')])},
               {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
               {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
               {'call': 'inotify_add_watch',
                'reason': set([('fd', 'flags')])},
               {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
               {'call': 'splice', 'reason': set([('fd', 'flags')])},
               {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv', 'reason': set([('fd', 'flags')])},
               {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
```



```
               {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
               {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
               {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
               {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
               {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
               {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
               {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
               {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
               {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
               {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
               {'call': 'listen', 'reason': set([('fd', 'flags')])},
               {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
               {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
               {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
               {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
               {'call': 'llseek', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
               {'call': 'readv', 'reason': set([('fd', 'flags')])},
               {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
               {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
               {'call': 'write', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
               {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
               {'call': 'bind', 'reason': set([('fd', 'flags')])},
               {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'epoll_wait': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'rt_sigtimedwait',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                {'call': 'sched_getaffinity',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'writev', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                {'call': 'sched_setparam',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                {'call': 'ioprio_set',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'read', 'reason': set([('fd', 'flags')])},
                {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
                {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'utime', 'reason': set([('fd', 'flags')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
```

```
654                 {'call': 'bpf', 'reason': set([('fd', 'flags')])},
655                 {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
656                 {'call': 'semtimedop',
657                  'reason': set([('mm_segment_t', 'seg')])},
658                 {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
659                 {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
660                 {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
661                 {'call': 'sendto', 'reason': set([('fd', 'flags')])},
662                 {'call': 'sched_rr_get_interval',
663                  'reason': set([('mm_segment_t', 'seg')])},
664                 {'call': 'tee', 'reason': set([('fd', 'flags')])},
665                 {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
666                 {'call': 'lseek', 'reason': set([('fd', 'flags')])},
667                 {'call': 'connect', 'reason': set([('fd', 'flags')])},
668                 {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
669                 {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
670                 {'call': 'flock', 'reason': set([('fd', 'flags')])},
671                 {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
672                 {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
673                 {'call': 'rt_sigprocmask',
674                  'reason': set([('mm_segment_t', 'seg')])},
675                 {'call': 'accept4', 'reason': set([('fd', 'flags')])},
676                 {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
677                 {'call': 'sigaltstack',
678                  'reason': set([('mm_segment_t', 'seg')])},
679                 {'call': 'sched_setattr',
680                  'reason': set([('mm_segment_t', 'seg')])},
681                 {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
682                 {'call': 'inotify_rm_watch',
683                  'reason': set([('fd', 'flags')])},
684                 {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
685                 {'call': 'migrate_pages',
686                  'reason': set([('mm_segment_t', 'seg')])},
687                 {'call': 'getitimer',
688                  'reason': set([('mm_segment_t', 'seg')])},
689                 {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
690                 {'call': 'inotify_add_watch',
691                  'reason': set([('fd', 'flags')])},
692                 {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
693                 {'call': 'splice', 'reason': set([('fd', 'flags')])},
694                 {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
695                 {'call': 'preadv', 'reason': set([('fd', 'flags')])},
696                 {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
697                 {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
698                 {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
699                 {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
700                 {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
701                 {'call': 'prlimit64',
702                  'reason': set([('mm_segment_t', 'seg')])},
703                 {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
704                 {'call': 'perf_event_open',
705                  'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
706                 {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
707                 {'call': 'rt_sigaction',
708                  'reason': set([('mm_segment_t', 'seg')])},
709                 {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
710                 {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
711                 {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
712                 {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
713                 {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
714                 {'call': 'getpriority',
715                  'reason': set([('mm_segment_t', 'seg')])},
716                 {'call': 'sigaction',
717                  'reason': set([('mm_segment_t', 'seg')])},
718                 {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
719                 {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
```



```
720                 {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
721                 {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
722                 {'call': 'listen', 'reason': set([('fd', 'flags')])},
723                 {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
724                 {'call': 'get_robust_list',
725                  'reason': set([('mm_segment_t', 'seg')])},
726                 {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
727                 {'call': 'mq_timedsend',
728                  'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
729                 {'call': 'sched_getscheduler',
730                  'reason': set([('mm_segment_t', 'seg')])},
731                 {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
732                 {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
733                 {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
734                 {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
735                 {'call': 'sched_getattr',
736                  'reason': set([('mm_segment_t', 'seg')])},
737                 {'call': 'getrusage',
738                  'reason': set([('mm_segment_t', 'seg')])},
739                 {'call': 'sched_setscheduler',
740                  'reason': set([('mm_segment_t', 'seg')])},
741                 {'call': 'setitimer',
742                  'reason': set([('mm_segment_t', 'seg')])},
743                 {'call': 'ioprio_get',
744                  'reason': set([('mm_segment_t', 'seg')])},
745                 {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
746                 {'call': 'llseek', 'reason': set([('fd', 'flags')])},
747                 {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
748                 {'call': 'readv', 'reason': set([('fd', 'flags')])},
749                 {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
750                 {'call': 'move_pages',
751                  'reason': set([('mm_segment_t', 'seg')])},
752                 {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
753                 {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
754                 {'call': 'write', 'reason': set([('fd', 'flags')])},
755                 {'call': 'setpriority',
756                  'reason': set([('mm_segment_t', 'seg')])},
757                 {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
758                 {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
759                 {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
760                 {'call': 'sched_getparam',
761                  'reason': set([('mm_segment_t', 'seg')])},
762                 {'call': 'bind', 'reason': set([('fd', 'flags')])},
763                 {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
764                 {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
765  'fadvise64_64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
766                   {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
767                   {'call': 'mq_unlink',
768                    'reason': set([('inode', 'i_flags')])},
769                   {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
770                   {'call': 'swapoff', 'reason': set([('inode', 'i_flags')])},
771                   {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
772                   {'call': 'readahead', 'reason': set([('fd', 'flags')])},
773                   {'call': 'getdents', 'reason': set([('fd', 'flags')])},
774                   {'call': 'writev', 'reason': set([('fd', 'flags')])},
775                   {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
776                   {'call': 'fchmod',
777                    'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
778                   {'call': 'pread64', 'reason': set([('fd', 'flags')])},
779                   {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
780                   {'call': 'memfd_create',
781                    'reason': set([('inode', 'i_flags')])},
782                   {'call': 'readlinkat',
783                    'reason': set([('inode', 'i_flags')])},
784                   {'call': 'read', 'reason': set([('fd', 'flags')])},
785                   {'call': 'fchown',
```

```
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'utime', 'reason': set([('fd', 'flags')])},
                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                {'call': 'tee', 'reason': set([('fd', 'flags')])},
                {'call': 'sync_file_range',
                 'reason': set([('fd', 'flags')])},
                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                {'call': 'connect', 'reason': set([('fd', 'flags')])},
                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
                {'call': 'flock', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                {'call': 'uselib', 'reason': set([('inode', 'i_flags')])},
                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
                {'call': 'inotify_rm_watch',
                 'reason': set([('fd', 'flags')])},
                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                {'call': 'fchmodat', 'reason': set([('inode', 'i_flags')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                {'call': 'splice', 'reason': set([('fd', 'flags')])},
                {'call': 'ftruncate',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                {'call': 'ioctl',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                {'call': 'perf_event_open',
                 'reason': set([('fd', 'flags')])},
                {'call': 'linkat', 'reason': set([('inode', 'i_flags')])},
                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                {'call': 'unlink', 'reason': set([('inode', 'i_flags')])},
                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_getsetattr',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'faccessat',
                 'reason': set([('inode', 'i_flags')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                {'call': 'listen', 'reason': set([('fd', 'flags')])},
                {'call': 'copy_file_range',
                 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedsend',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                {'call': 'swapon', 'reason': set([('inode', 'i_flags')])},
                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                {'call': 'fchownat', 'reason': set([('inode', 'i_flags')])},
                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
```



```
                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'readv', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                {'call': 'write', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_notify',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'sendfile',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])},
                {'call': 'unlinkat', 'reason': set([('inode', 'i_flags')])},
                {'call': 'bind', 'reason': set([('fd', 'flags')])},
                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile64',
                 'reason': set([('fd', 'flags'), ('inode', 'i_flags')])}],
 'fallocate': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
               {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
               {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
               {'call': 'readahead', 'reason': set([('fd', 'flags')])},
               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
               {'call': 'writev', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
               {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
               {'call': 'pread64', 'reason': set([('fd', 'flags')])},
               {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
               {'call': 'read', 'reason': set([('fd', 'flags')])},
               {'call': 'fchown', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
               {'call': 'utime', 'reason': set([('fd', 'flags')])},
               {'call': 'fsync', 'reason': set([('fd', 'flags')])},
               {'call': 'bpf', 'reason': set([('fd', 'flags')])},
               {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
               {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'sendto', 'reason': set([('fd', 'flags')])},
               {'call': 'tee', 'reason': set([('fd', 'flags')])},
               {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
               {'call': 'lseek', 'reason': set([('fd', 'flags')])},
               {'call': 'connect', 'reason': set([('fd', 'flags')])},
               {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
               {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
               {'call': 'flock', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
               {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
               {'call': 'accept4', 'reason': set([('fd', 'flags')])},
               {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
               {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
               {'call': 'inotify_add_watch',
                'reason': set([('fd', 'flags')])},
               {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
               {'call': 'splice', 'reason': set([('fd', 'flags')])},
               {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv', 'reason': set([('fd', 'flags')])},
               {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
               {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
               {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
               {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
               {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
               {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
               {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
               {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
               {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
```

```
                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                {'call': 'listen', 'reason': set([('fd', 'flags')])},
                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'readv', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                {'call': 'write', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                {'call': 'bind', 'reason': set([('fd', 'flags')])},
                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fchdir': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
            {'call': 'flock', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
```



```
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fchmod': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
            {'call': 'flock', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fchown': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
            {'call': 'flock', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
```



```
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fcntl': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
           {'call': 'fadvise64_64',
            'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
           {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
           {'call': 'readahead', 'reason': set([('fd', 'flags')])},
           {'call': 'getdents', 'reason': set([('fd', 'flags')])},
           {'call': 'writev', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
           {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
           {'call': 'pread64', 'reason': set([('fd', 'flags')])},
           {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
           {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
           {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
           {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
           {'call': 'read', 'reason': set([('fd', 'flags')])},
           {'call': 'fchown', 'reason': set([('fd', 'flags')])},
           {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
           {'call': 'utime', 'reason': set([('fd', 'flags')])},
           {'call': 'fsync', 'reason': set([('fd', 'flags')])},
           {'call': 'bpf', 'reason': set([('fd', 'flags')])},
           {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
           {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
           {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
           {'call': 'sendto', 'reason': set([('fd', 'flags')])},
           {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
           {'call': 'tee', 'reason': set([('fd', 'flags')])},
           {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
           {'call': 'lseek', 'reason': set([('fd', 'flags')])},
           {'call': 'connect', 'reason': set([('fd', 'flags')])},
           {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
           {'call': 'epoll_ctl',
            'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
```

```
1182            {'call': 'flock',
1183             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1184            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
1185            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
1186            {'call': 'openat', 'reason': set([('file', 'f_mode')])},
1187            {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
1188            {'call': 'accept4',
1189             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1190            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
1191            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
1192            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
1193            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
1194            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
1195            {'call': 'splice', 'reason': set([('fd', 'flags')])},
1196            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
1197            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
1198            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
1199            {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
1200            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
1201            {'call': 'socket', 'reason': set([('file', 'f_mode')])},
1202            {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
1203            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
1204            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
1205            {'call': 'perf_event_open',
1206             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1207            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
1208            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
1209            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
1210            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
1211            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
1212            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
1213            {'call': 'open', 'reason': set([('file', 'f_mode')])},
1214            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
1215            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
1216            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
1217            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
1218            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
1219            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
1220            {'call': 'listen', 'reason': set([('fd', 'flags')])},
1221            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
1222            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
1223            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
1224            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
1225            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
1226            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
1227            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
1228            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
1229            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
1230            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
1231            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
1232            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
1233            {'call': 'readv', 'reason': set([('fd', 'flags')])},
1234            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
1235            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
1236            {'call': 'write', 'reason': set([('fd', 'flags')])},
1237            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
1238            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
1239            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
1240            {'call': 'msync', 'reason': set([('file', 'f_mode')])},
1241            {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
1242            {'call': 'bind', 'reason': set([('fd', 'flags')])},
1243            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
1244            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
1245  'fcntl64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
1246              {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
1247              {'call': 'fadvise64_64',
```



```
1248               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1249              {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
1250              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
1251              {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
1252              {'call': 'readahead', 'reason': set([('fd', 'flags')])},
1253              {'call': 'getdents', 'reason': set([('fd', 'flags')])},
1254              {'call': 'writev', 'reason': set([('fd', 'flags')])},
1255              {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
1256              {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
1257              {'call': 'pread64', 'reason': set([('fd', 'flags')])},
1258              {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
1259              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
1260              {'call': 'remap_file_pages',
1261               'reason': set([('file', 'f_mode')])},
1262              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
1263              {'call': 'read', 'reason': set([('fd', 'flags')])},
1264              {'call': 'fchown', 'reason': set([('fd', 'flags')])},
1265              {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
1266              {'call': 'utime', 'reason': set([('fd', 'flags')])},
1267              {'call': 'fsync', 'reason': set([('fd', 'flags')])},
1268              {'call': 'bpf', 'reason': set([('fd', 'flags')])},
1269              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
1270              {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
1271              {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
1272              {'call': 'sendto', 'reason': set([('fd', 'flags')])},
1273              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
1274              {'call': 'tee', 'reason': set([('fd', 'flags')])},
1275              {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
1276              {'call': 'lseek', 'reason': set([('fd', 'flags')])},
1277              {'call': 'connect', 'reason': set([('fd', 'flags')])},
1278              {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
1279              {'call': 'epoll_ctl',
1280               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1281              {'call': 'flock',
1282               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1283              {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
1284              {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
1285              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
1286              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
1287              {'call': 'accept4',
1288               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1289              {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
1290              {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
1291              {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
1292              {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
1293              {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
1294              {'call': 'splice', 'reason': set([('fd', 'flags')])},
1295              {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
1296              {'call': 'preadv', 'reason': set([('fd', 'flags')])},
1297              {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
1298              {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
1299              {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
1300              {'call': 'socket', 'reason': set([('file', 'f_mode')])},
1301              {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
1302              {'call': 'fcntl',
1303               'reason': set([('fd', 'flags'),
1304                              ('flock', 'l_len'),
1305                              ('flock', 'l_start')])},
1306              {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
1307              {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
1308              {'call': 'perf_event_open',
1309               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
1310              {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
1311              {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
1312              {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
1313              {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
            {'call': 'open', 'reason': set([('file', 'f_mode')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'msync', 'reason': set([('file', 'f_mode')])},
            {'call': 'open_by_handle_at',
             'reason': set([('file', 'f_mode')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'fgetxattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
               {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
               {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
               {'call': 'readahead', 'reason': set([('fd', 'flags')])},
               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
               {'call': 'writev', 'reason': set([('fd', 'flags')])},
               {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
               {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
               {'call': 'pread64', 'reason': set([('fd', 'flags')])},
               {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
               {'call': 'read', 'reason': set([('fd', 'flags')])},
               {'call': 'fchown', 'reason': set([('fd', 'flags')])},
               {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
               {'call': 'utime', 'reason': set([('fd', 'flags')])},
               {'call': 'fsync', 'reason': set([('fd', 'flags')])},
               {'call': 'bpf', 'reason': set([('fd', 'flags')])},
               {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
               {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
               {'call': 'sendto', 'reason': set([('fd', 'flags')])},
               {'call': 'tee', 'reason': set([('fd', 'flags')])},
               {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
               {'call': 'lseek', 'reason': set([('fd', 'flags')])},
               {'call': 'connect', 'reason': set([('fd', 'flags')])},
               {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
               {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
               {'call': 'flock', 'reason': set([('fd', 'flags')])},
               {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
               {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
               {'call': 'accept4', 'reason': set([('fd', 'flags')])},
               {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
```



```
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch',
             'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'finit_module': [{'call': 'delete_module',
                   'reason': set([('module', 'args'),
                                  ('module', 'num_kp'),
                                  ('module_layout', 'base'),
                                  ('module_layout', 'size')])},
                  {'call': 'init_module',
                   'reason': set([('load_info', 'num_debug'),
                                  ('module', 'args'),
                                  ('module', 'num_kp'),
                                  ('module_layout', 'base'),
                                  ('module_layout', 'size')])}],
 'flistxattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                {'call': 'writev', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                {'call': 'read', 'reason': set([('fd', 'flags')])},
                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
                {'call': 'utime', 'reason': set([('fd', 'flags')])},
```

```
                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                {'call': 'tee', 'reason': set([('fd', 'flags')])},
                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                {'call': 'connect', 'reason': set([('fd', 'flags')])},
                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
                {'call': 'flock', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
                {'call': 'inotify_rm_watch',
                 'reason': set([('fd', 'flags')])},
                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('fd', 'flags')])},
                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                {'call': 'splice', 'reason': set([('fd', 'flags')])},
                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                {'call': 'listen', 'reason': set([('fd', 'flags')])},
                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'readv', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                {'call': 'write', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                {'call': 'bind', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'flock': [{'call': 'syncfs',
            'reason': set([('fd', 'flags'), ('super_block', 's_flags')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
           {'call': 'fadvise64_64',
            'reason': set([('fd', 'flags'),
                           ('file', 'f_mode'),
                           ('super_block', 's_flags')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
           {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
```



```
           {'call': 'readahead', 'reason': set([('fd', 'flags')])},
           {'call': 'getdents', 'reason': set([('fd', 'flags')])},
           {'call': 'writev', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
           {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
           {'call': 'pread64', 'reason': set([('fd', 'flags')])},
           {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
           {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
           {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
           {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
           {'call': 'read', 'reason': set([('fd', 'flags')])},
           {'call': 'fchown', 'reason': set([('fd', 'flags')])},
           {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
           {'call': 'utime', 'reason': set([('fd', 'flags')])},
           {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
           {'call': 'fsync', 'reason': set([('fd', 'flags')])},
           {'call': 'bpf', 'reason': set([('fd', 'flags')])},
           {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
           {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
           {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
           {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
           {'call': 'sendto', 'reason': set([('fd', 'flags')])},
           {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
           {'call': 'tee', 'reason': set([('fd', 'flags')])},
           {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
           {'call': 'lseek', 'reason': set([('fd', 'flags')])},
           {'call': 'connect', 'reason': set([('fd', 'flags')])},
           {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
           {'call': 'epoll_ctl',
            'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
           {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
           {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
           {'call': 'openat', 'reason': set([('file', 'f_mode')])},
           {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
           {'call': 'accept4',
            'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
           {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
           {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
           {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
           {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
           {'call': 'splice', 'reason': set([('fd', 'flags')])},
           {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv', 'reason': set([('fd', 'flags')])},
           {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
           {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
           {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
           {'call': 'socket', 'reason': set([('file', 'f_mode')])},
           {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
           {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
           {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
           {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
           {'call': 'perf_event_open',
            'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
           {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
           {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
           {'call': 'quotactl', 'reason': set([('super_block', 's_flags')])},
           {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
           {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
           {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
           {'call': 'acct', 'reason': set([('file', 'f_mode')])},
           {'call': 'open', 'reason': set([('file', 'f_mode')])},
           {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
           {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
           {'call': 'dup', 'reason': set([('file', 'f_mode')])},
           {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
```

```
1578            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
1579            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
1580            {'call': 'listen', 'reason': set([('fd', 'flags')])},
1581            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
1582            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
1583            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
1584            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
1585            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
1586            {'call': 'swapon',
1587             'reason': set([('file', 'f_mode'), ('super_block', 's_flags')])},
1588            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
1589            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
1590            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
1591            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
1592            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
1593            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
1594            {'call': 'readv', 'reason': set([('fd', 'flags')])},
1595            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
1596            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
1597            {'call': 'write', 'reason': set([('fd', 'flags')])},
1598            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
1599            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
1600            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
1601            {'call': 'msync', 'reason': set([('file', 'f_mode')])},
1602            {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
1603            {'call': 'bind', 'reason': set([('fd', 'flags')])},
1604            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
1605            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
1606  'fremovexattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
1607                   {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
1608                   {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
1609                   {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
1610                   {'call': 'readahead', 'reason': set([('fd', 'flags')])},
1611                   {'call': 'getdents', 'reason': set([('fd', 'flags')])},
1612                   {'call': 'writev', 'reason': set([('fd', 'flags')])},
1613                   {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
1614                   {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
1615                   {'call': 'pread64', 'reason': set([('fd', 'flags')])},
1616                   {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
1617                   {'call': 'read', 'reason': set([('fd', 'flags')])},
1618                   {'call': 'fchown', 'reason': set([('fd', 'flags')])},
1619                   {'call': 'mq_timedreceive',
1620                    'reason': set([('fd', 'flags')])},
1621                   {'call': 'utime', 'reason': set([('fd', 'flags')])},
1622                   {'call': 'fsync', 'reason': set([('fd', 'flags')])},
1623                   {'call': 'bpf', 'reason': set([('fd', 'flags')])},
1624                   {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
1625                   {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
1626                   {'call': 'sendto', 'reason': set([('fd', 'flags')])},
1627                   {'call': 'tee', 'reason': set([('fd', 'flags')])},
1628                   {'call': 'sync_file_range',
1629                    'reason': set([('fd', 'flags')])},
1630                   {'call': 'lseek', 'reason': set([('fd', 'flags')])},
1631                   {'call': 'connect', 'reason': set([('fd', 'flags')])},
1632                   {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
1633                   {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
1634                   {'call': 'flock', 'reason': set([('fd', 'flags')])},
1635                   {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
1636                   {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
1637                   {'call': 'accept4', 'reason': set([('fd', 'flags')])},
1638                   {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
1639                   {'call': 'inotify_rm_watch',
1640                    'reason': set([('fd', 'flags')])},
1641                   {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
1642                   {'call': 'inotify_add_watch',
1643                    'reason': set([('fd', 'flags')])},
```



```
1644                   {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
1645                   {'call': 'splice', 'reason': set([('fd', 'flags')])},
1646                   {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
1647                   {'call': 'preadv', 'reason': set([('fd', 'flags')])},
1648                   {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
1649                   {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
1650                   {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
1651                   {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
1652                   {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
1653                   {'call': 'perf_event_open',
1654                    'reason': set([('fd', 'flags')])},
1655                   {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
1656                   {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
1657                   {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
1658                   {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
1659                   {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
1660                   {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
1661                   {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
1662                   {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
1663                   {'call': 'listen', 'reason': set([('fd', 'flags')])},
1664                   {'call': 'copy_file_range',
1665                    'reason': set([('fd', 'flags')])},
1666                   {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
1667                   {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
1668                   {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
1669                   {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
1670                   {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
1671                   {'call': 'llseek', 'reason': set([('fd', 'flags')])},
1672                   {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
1673                   {'call': 'readv', 'reason': set([('fd', 'flags')])},
1674                   {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
1675                   {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
1676                   {'call': 'write', 'reason': set([('fd', 'flags')])},
1677                   {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
1678                   {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
1679                   {'call': 'bind', 'reason': set([('fd', 'flags')])},
1680                   {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
1681                   {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
1682  'fsetxattr': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
1683                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
1684                {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
1685                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
1686                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
1687                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
1688                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
1689                {'call': 'writev', 'reason': set([('fd', 'flags')])},
1690                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
1691                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
1692                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
1693                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
1694                {'call': 'read', 'reason': set([('fd', 'flags')])},
1695                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
1696                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
1697                {'call': 'utime', 'reason': set([('fd', 'flags')])},
1698                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
1699                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
1700                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
1701                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
1702                {'call': 'tee', 'reason': set([('fd', 'flags')])},
1703                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
1704                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
1705                {'call': 'connect', 'reason': set([('fd', 'flags')])},
1706                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
1707                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
1708                {'call': 'flock', 'reason': set([('fd', 'flags')])},
1709                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
```

```
1710                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
1711                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
1712                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
1713                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
1714                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
1715                {'call': 'inotify_add_watch',
1716                 'reason': set([('fd', 'flags')])},
1717                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
1718                {'call': 'splice', 'reason': set([('fd', 'flags')])},
1719                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
1720                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
1721                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
1722                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
1723                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
1724                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
1725                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
1726                {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
1727                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
1728                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
1729                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
1730                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
1731                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
1732                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
1733                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
1734                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
1735                {'call': 'listen', 'reason': set([('fd', 'flags')])},
1736                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
1737                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
1738                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
1739                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
1740                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
1741                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
1742                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
1743                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
1744                {'call': 'readv', 'reason': set([('fd', 'flags')])},
1745                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
1746                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
1747                {'call': 'write', 'reason': set([('fd', 'flags')])},
1748                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
1749                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
1750                {'call': 'bind', 'reason': set([('fd', 'flags')])},
1751                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
1752                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
1753  'fstat': [{'call': 'lstat',
1754             'reason': set([('__old_kernel_stat', 'st_ino'),
1755                            ('__old_kernel_stat', 'st_nlink')])},
1756            {'call': 'stat',
1757             'reason': set([('__old_kernel_stat', 'st_ino'),
1758                            ('__old_kernel_stat', 'st_nlink')])}],
1759  'fstatfs': [{'call': 'ustat',
1760               'reason': set([('kstatfs', 'f_ffree'),
1761                              ('kstatfs', 'f_files')])},
1762              {'call': 'statfs',
1763               'reason': set([('kstatfs', 'f_ffree'),
1764                              ('kstatfs', 'f_files')])},
1765              {'call': 'fstatfs64',
1766               'reason': set([('kstatfs', 'f_ffree'),
1767                              ('kstatfs', 'f_files')])},
1768              {'call': 'statfs64',
1769               'reason': set([('kstatfs', 'f_ffree'),
1770                              ('kstatfs', 'f_files')])}],
1771  'fstatfs64': [{'call': 'ustat',
1772                 'reason': set([('kstatfs', 'f_ffree'),
1773                                ('kstatfs', 'f_files')])},
1774                {'call': 'fstatfs',
1775                 'reason': set([('kstatfs', 'f_ffree'),
```



```
1776                                ('kstatfs', 'f_files')])},
1777               {'call': 'statfs',
1778                'reason': set([('kstatfs', 'f_ffree'),
1779                               ('kstatfs', 'f_files')])},
1780               {'call': 'statfs64',
1781                'reason': set([('kstatfs', 'f_ffree'),
1782                               ('kstatfs', 'f_files')])}],
1783  'ftruncate': [{'call': 'fadvise64_64',
1784                 'reason': set([('file', 'f_mode'), ('inode', 'i_flags')])},
1785                {'call': 'mq_unlink', 'reason': set([('inode', 'i_flags')])},
1786                {'call': 'swapoff',
1787                 'reason': set([('file', 'f_flags'),
1788                                ('file', 'f_mode'),
1789                                ('inode', 'i_flags')])},
1790                {'call': 'fchmod', 'reason': set([('inode', 'i_flags')])},
1791                {'call': 'memfd_create',
1792                 'reason': set([('file', 'f_flags'),
1793                                ('file', 'f_mode'),
1794                                ('inode', 'i_flags')])},
1795                {'call': 'remap_file_pages',
1796                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1797                {'call': 'dup3',
1798                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1799                {'call': 'readlinkat', 'reason': set([('inode', 'i_flags')])},
1800                {'call': 'fchown', 'reason': set([('inode', 'i_flags')])},
1801                {'call': 'mq_timedreceive',
1802                 'reason': set([('inode', 'i_flags')])},
1803                {'call': 'socketpair',
1804                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1805                {'call': 'epoll_create1',
1806                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1807                {'call': 'epoll_ctl',
1808                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1809                {'call': 'flock',
1810                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1811                {'call': 'openat',
1812                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1813                {'call': 'uselib',
1814                 'reason': set([('file', 'f_flags'),
1815                                ('file', 'f_mode'),
1816                                ('inode', 'i_flags')])},
1817                {'call': 'accept4',
1818                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1819                {'call': 'fchmodat', 'reason': set([('inode', 'i_flags')])},
1820                {'call': 'inotify_add_watch',
1821                 'reason': set([('inode', 'i_flags')])},
1822                {'call': 'shmat',
1823                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1824                {'call': 'socket',
1825                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1826                {'call': 'pipe2',
1827                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1828                {'call': 'ioctl', 'reason': set([('inode', 'i_flags')])},
1829                {'call': 'perf_event_open',
1830                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1831                {'call': 'linkat', 'reason': set([('inode', 'i_flags')])},
1832                {'call': 'shmdt',
1833                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1834                {'call': 'acct',
1835                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1836                {'call': 'open',
1837                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
1838                {'call': 'unlink', 'reason': set([('inode', 'i_flags')])},
1839                {'call': 'mq_getsetattr',
1840                 'reason': set([('file', 'f_flags'), ('inode', 'i_flags')])},
1841                {'call': 'faccessat', 'reason': set([('inode', 'i_flags')])},
```

```
                {'call': 'dup',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'setns',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'mq_timedsend',
                 'reason': set([('inode', 'i_flags')])},
                {'call': 'shmctl',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'swapon',
                 'reason': set([('file', 'f_flags'),
                                ('file', 'f_mode'),
                                ('inode', 'i_flags')])},
                {'call': 'fchownat', 'reason': set([('inode', 'i_flags')])},
                {'call': 'eventfd2',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'mmap_pgoff',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'mq_notify', 'reason': set([('inode', 'i_flags')])},
                {'call': 'sendfile', 'reason': set([('inode', 'i_flags')])},
                {'call': 'mq_open',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'msync',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'unlinkat', 'reason': set([('inode', 'i_flags')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
                {'call': 'sendfile64', 'reason': set([('inode', 'i_flags')])}],
 'futex': [{'call': 'rt_sigtimedwait',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'fadvise64_64',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'mq_unlink',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'swapoff',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'fchmod',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'memfd_create',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'readlinkat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'io_getevents',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'fchown',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'mq_timedreceive',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'utime',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'semtimedop',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'settimeofday',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'sched_rr_get_interval',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'timerfd_gettime',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'pselect6',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'uselib',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'fchmodat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'inotify_add_watch',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'timer_settime',
```



```
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'ftruncate',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'timer_gettime',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'ioctl',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'linkat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'stime',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'futimesat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'poll',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'select',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'unlink',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'nanosleep',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'mq_getsetattr',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'faccessat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'mq_timedsend',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'swapon',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'epoll_wait',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'fchownat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'timerfd_settime',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'mq_notify',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'sendfile',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'clock_nanosleep',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'unlinkat',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'recvmmsg',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'sendfile64',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
           {'call': 'ppoll',
            'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])}],
 'futimesat': [{'call': 'rt_sigtimedwait',
                'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'fadvise64_64',
                'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'mq_unlink',
                'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'memfd_create',
                'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'readlinkat',
                'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'io_getevents',
                'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
               {'call': 'mq_timedreceive',
                'reason': set([('timespec', 'tv_nsec')])},
```

```
                {'call': 'utime', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'semtimedop',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'settimeofday',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'timerfd_gettime',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'adjtimex', 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'waitid', 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'getitimer', 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'timer_settime',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'ftruncate',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'timer_gettime',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'select',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'nanosleep',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'mq_getsetattr',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'faccessat',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'mq_timedsend',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'wait4', 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'epoll_wait',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'getrusage', 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'timerfd_settime',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'setitimer', 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'mq_notify',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'clock_nanosleep',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'clock_adjtime',
                 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'alarm', 'reason': set([('timeval', 'tv_usec')])},
                {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'sendfile64',
                 'reason': set([('timespec', 'tv_nsec')])},
                {'call': 'ppoll',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timeval', 'tv_usec')])}],
'get_mempolicy': [{'call': 'keyctl',
```



```
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'rt_sigtimedwait',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'msgrcv',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'kill',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'sched_getaffinity',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'sched_setparam',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'ioprio_set',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'getppid',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'capget',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'sched_setaffinity',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'signal',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'semtimedop',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'umount',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'rt_sigprocmask',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'setsid',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'sigaltstack',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'sched_setattr',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'migrate_pages',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'getitimer',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'setpgid',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'getsid',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'prlimit64',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'set_mempolicy',
                 'reason': set([('mempolicy', 'mode'),
                                ('task_struct', 'il_prev')])},
                {'call': 'perf_event_open',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'rt_sigaction',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'getpgid',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'getpriority',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'sigaction',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'setns',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'fork',
                 'reason': set([('task_struct', 'il_prev')])},
                {'call': 'get_robust_list',
                 'reason': set([('task_struct', 'il_prev')])},
```

```
                    {'call': 'mq_timedsend',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'ptrace',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'sched_getattr',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'getrusage',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'setitimer',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'ioprio_get',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'vfork',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'mbind', 'reason': set([('mempolicy', 'mode')])},
                    {'call': 'prctl',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'move_pages',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'setpriority',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'clone',
                     'reason': set([('task_struct', 'il_prev')])},
                    {'call': 'sched_getparam',
                     'reason': set([('task_struct', 'il_prev')])}],
 'getdents': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
              {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
              {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
              {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
              {'call': 'readahead', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_getaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'writev', 'reason': set([('fd', 'flags')])},
              {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_setparam',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
              {'call': 'pread64', 'reason': set([('fd', 'flags')])},
              {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
              {'call': 'ioprio_set',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'read', 'reason': set([('fd', 'flags')])},
              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchown', 'reason': set([('fd', 'flags')])},
              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_timedreceive',
               'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'utime', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_setaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fsync', 'reason': set([('fd', 'flags')])},
              {'call': 'bpf', 'reason': set([('fd', 'flags')])},
              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'semtimedop',
```



```
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
              {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
              {'call': 'sendto', 'reason': set([('fd', 'flags')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'tee', 'reason': set([('fd', 'flags')])},
              {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
              {'call': 'lseek', 'reason': set([('fd', 'flags')])},
              {'call': 'connect', 'reason': set([('fd', 'flags')])},
              {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
              {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
              {'call': 'flock', 'reason': set([('fd', 'flags')])},
              {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
              {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'accept4', 'reason': set([('fd', 'flags')])},
              {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sigaltstack',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sched_setattr',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
              {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
              {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
              {'call': 'migrate_pages',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
              {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
              {'call': 'splice', 'reason': set([('fd', 'flags')])},
              {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
              {'call': 'preadv', 'reason': set([('fd', 'flags')])},
              {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
              {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
              {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
              {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
              {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
              {'call': 'perf_event_open',
               'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
              {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
              {'call': 'rt_sigaction',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
              {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
              {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
              {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
              {'call': 'getpriority',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
              {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
              {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
              {'call': 'listen', 'reason': set([('fd', 'flags')])},
              {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'get_robust_list',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
              {'call': 'mq_timedsend',
```

```
2238                'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
2239               {'call': 'sched_getscheduler',
2240                'reason': set([('mm_segment_t', 'seg')])},
2241               {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
2242               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
2243               {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
2244               {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
2245               {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
2246               {'call': 'sched_getattr',
2247                'reason': set([('mm_segment_t', 'seg')])},
2248               {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
2249               {'call': 'sched_setscheduler',
2250                'reason': set([('mm_segment_t', 'seg')])},
2251               {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
2252               {'call': 'ioprio_get',
2253                'reason': set([('mm_segment_t', 'seg')])},
2254               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
2255               {'call': 'llseek', 'reason': set([('fd', 'flags')])},
2256               {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
2257               {'call': 'readv', 'reason': set([('fd', 'flags')])},
2258               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
2259               {'call': 'move_pages',
2260                'reason': set([('mm_segment_t', 'seg')])},
2261               {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
2262               {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
2263               {'call': 'write', 'reason': set([('fd', 'flags')])},
2264               {'call': 'setpriority',
2265                'reason': set([('mm_segment_t', 'seg')])},
2266               {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
2267               {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
2268               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
2269               {'call': 'sched_getparam',
2270                'reason': set([('mm_segment_t', 'seg')])},
2271               {'call': 'bind', 'reason': set([('fd', 'flags')])},
2272               {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
2273               {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
2274  'getdents64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
2275                 {'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
2276                 {'call': 'rt_sigtimedwait',
2277                  'reason': set([('mm_segment_t', 'seg')])},
2278                 {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
2279                 {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
2280                 {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
2281                 {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
2282                 {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
2283                 {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
2284                 {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
2285                 {'call': 'readahead', 'reason': set([('fd', 'flags')])},
2286                 {'call': 'getdents', 'reason': set([('fd', 'flags')])},
2287                 {'call': 'sched_getaffinity',
2288                  'reason': set([('mm_segment_t', 'seg')])},
2289                 {'call': 'writev', 'reason': set([('fd', 'flags')])},
2290                 {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
2291                 {'call': 'sched_setparam',
2292                  'reason': set([('mm_segment_t', 'seg')])},
2293                 {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
2294                 {'call': 'pread64', 'reason': set([('fd', 'flags')])},
2295                 {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
2296                 {'call': 'ioprio_set',
2297                  'reason': set([('mm_segment_t', 'seg')])},
2298                 {'call': 'read', 'reason': set([('fd', 'flags')])},
2299                 {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
2300                 {'call': 'fchown', 'reason': set([('fd', 'flags')])},
2301                 {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
2302                 {'call': 'mq_timedreceive',
2303                  'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
```



```
2304                {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
2305                {'call': 'utime', 'reason': set([('fd', 'flags')])},
2306                {'call': 'sched_setaffinity',
2307                 'reason': set([('mm_segment_t', 'seg')])},
2308                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
2309                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
2310                {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
2311                {'call': 'semtimedop',
2312                 'reason': set([('mm_segment_t', 'seg')])},
2313                {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
2314                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
2315                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
2316                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
2317                {'call': 'sched_rr_get_interval',
2318                 'reason': set([('mm_segment_t', 'seg')])},
2319                {'call': 'tee', 'reason': set([('fd', 'flags')])},
2320                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
2321                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
2322                {'call': 'connect', 'reason': set([('fd', 'flags')])},
2323                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
2324                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
2325                {'call': 'flock', 'reason': set([('fd', 'flags')])},
2326                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
2327                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
2328                {'call': 'rt_sigprocmask',
2329                 'reason': set([('mm_segment_t', 'seg')])},
2330                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
2331                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
2332                {'call': 'sigaltstack',
2333                 'reason': set([('mm_segment_t', 'seg')])},
2334                {'call': 'sched_setattr',
2335                 'reason': set([('mm_segment_t', 'seg')])},
2336                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
2337                {'call': 'inotify_rm_watch',
2338                 'reason': set([('fd', 'flags')])},
2339                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
2340                {'call': 'migrate_pages',
2341                 'reason': set([('mm_segment_t', 'seg')])},
2342                {'call': 'getitimer',
2343                 'reason': set([('mm_segment_t', 'seg')])},
2344                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
2345                {'call': 'inotify_add_watch',
2346                 'reason': set([('fd', 'flags')])},
2347                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
2348                {'call': 'splice', 'reason': set([('fd', 'flags')])},
2349                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
2350                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
2351                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
2352                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
2353                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
2354                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
2355                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
2356                {'call': 'prlimit64',
2357                 'reason': set([('mm_segment_t', 'seg')])},
2358                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
2359                {'call': 'perf_event_open',
2360                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
2361                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
2362                {'call': 'rt_sigaction',
2363                 'reason': set([('mm_segment_t', 'seg')])},
2364                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
2365                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
2366                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
2367                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
2368                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
2369                {'call': 'getpriority',
```

```
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'sigaction',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'listen', 'reason': set([('fd', 'flags')])},
                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'get_robust_list',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
                {'call': 'mq_timedsend',
                 'reason': set([('fd', 'flags'), ('mm_segment_t', 'seg')])},
                {'call': 'sched_getscheduler',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                {'call': 'sched_getattr',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'getrusage',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'sched_setscheduler',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'setitimer',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'ioprio_get',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                {'call': 'readv', 'reason': set([('fd', 'flags')])},
                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'move_pages',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                {'call': 'write', 'reason': set([('fd', 'flags')])},
                {'call': 'setpriority',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'sched_getparam',
                 'reason': set([('mm_segment_t', 'seg')])},
                {'call': 'bind', 'reason': set([('fd', 'flags')])},
                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'getgroups16': [{'call': 'setgroups16',
                  'reason': set([('group_info', 'ngroups')])},
                 {'call': 'setgroups',
                  'reason': set([('group_info', 'ngroups')])}],
 'getitimer': [{'call': 'timer_create',
                'reason': set([('signal_struct', 'it_real_incr')])},
               {'call': 'exit_group',
                'reason': set([('signal_struct', 'it_real_incr')])},
               {'call': 'setitimer',
                'reason': set([('signal_struct', 'it_real_incr')])}],
 'getrlimit': [{'call': 'setrlimit',
                'reason': set([('rlimit', 'rlim_cur'),
                               ('rlimit', 'rlim_max')])},
               {'call': 'old_getrlimit',
                'reason': set([('rlimit', 'rlim_cur'),
                               ('rlimit', 'rlim_max')])},
```



```
               {'call': 'prlimit64',
                'reason': set([('rlimit', 'rlim_cur'),
                               ('rlimit', 'rlim_max')])}],
 'getrusage': [{'call': 'timer_create',
                'reason': set([('signal_struct', 'maxrss')])},
               {'call': 'exit_group',
                'reason': set([('signal_struct', 'maxrss')])}],
 'getsockopt': [{'call': 'accept4',
                 'reason': set([('proto_ops', 'compat_getsockopt')])}],
 'init_module': [{'call': 'delete_module',
                  'reason': set([('module', 'args'),
                                 ('module', 'num_kp'),
                                 ('module_layout', 'base'),
                                 ('module_layout', 'size')])},
                 {'call': 'finit_module',
                  'reason': set([('load_info', 'len'),
                                 ('load_info', 'num_debug'),
                                 ('module', 'args'),
                                 ('module', 'num_kp'),
                                 ('module_layout', 'base'),
                                 ('module_layout', 'size')])}],
 'inotify_add_watch': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                       {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                       {'call': 'fadvise64_64',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'pwritev64',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'fremovexattr',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'readahead',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                       {'call': 'writev', 'reason': set([('fd', 'flags')])},
                       {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                       {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                       {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                       {'call': 'signalfd4',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'read', 'reason': set([('fd', 'flags')])},
                       {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                       {'call': 'mq_timedreceive',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'utime', 'reason': set([('fd', 'flags')])},
                       {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                       {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                       {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                       {'call': 'fsetxattr',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                       {'call': 'tee', 'reason': set([('fd', 'flags')])},
                       {'call': 'sync_file_range',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                       {'call': 'connect', 'reason': set([('fd', 'flags')])},
                       {'call': 'getsockname',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'epoll_ctl',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'flock', 'reason': set([('fd', 'flags')])},
                       {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                       {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                       {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                       {'call': 'old_readdir',
                        'reason': set([('fd', 'flags')])},
                       {'call': 'inotify_rm_watch',
                        'reason': set([('fd', 'flags')])},
```

```
                     {'call': 'utimensat',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                     {'call': 'splice', 'reason': set([('fd', 'flags')])},
                     {'call': 'ftruncate',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                     {'call': 'getpeername',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'setsockopt',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                     {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                     {'call': 'perf_event_open',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev64v2',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'futimesat',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                     {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                     {'call': 'getsockopt',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'mq_getsetattr',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fdatasync',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'getdents64',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'listen', 'reason': set([('fd', 'flags')])},
                     {'call': 'copy_file_range',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'mq_timedsend',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fgetxattr',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                     {'call': 'fallocate',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'epoll_wait',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                     {'call': 'preadv64v2',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'readv', 'reason': set([('fd', 'flags')])},
                     {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                     {'call': 'fstatfs64',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'write', 'reason': set([('fd', 'flags')])},
                     {'call': 'mq_notify',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                     {'call': 'bind', 'reason': set([('fd', 'flags')])},
                     {'call': 'flistxattr',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'sendfile64',
                      'reason': set([('fd', 'flags')])}],
 'inotify_rm_watch': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
                      {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
                      {'call': 'fadvise64_64',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
                      {'call': 'fremovexattr',
                       'reason': set([('fd', 'flags')])},
                      {'call': 'readahead', 'reason': set([('fd', 'flags')])},
```



```
                     {'call': 'getdents', 'reason': set([('fd', 'flags')])},
                     {'call': 'writev', 'reason': set([('fd', 'flags')])},
                     {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
                     {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
                     {'call': 'pread64', 'reason': set([('fd', 'flags')])},
                     {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
                     {'call': 'read', 'reason': set([('fd', 'flags')])},
                     {'call': 'fchown', 'reason': set([('fd', 'flags')])},
                     {'call': 'mq_timedreceive',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'utime', 'reason': set([('fd', 'flags')])},
                     {'call': 'fsync', 'reason': set([('fd', 'flags')])},
                     {'call': 'bpf', 'reason': set([('fd', 'flags')])},
                     {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
                     {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
                     {'call': 'sendto', 'reason': set([('fd', 'flags')])},
                     {'call': 'tee', 'reason': set([('fd', 'flags')])},
                     {'call': 'sync_file_range',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'lseek', 'reason': set([('fd', 'flags')])},
                     {'call': 'connect', 'reason': set([('fd', 'flags')])},
                     {'call': 'getsockname',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
                     {'call': 'flock', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
                     {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                     {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                     {'call': 'old_readdir',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                     {'call': 'inotify_add_watch',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                     {'call': 'splice', 'reason': set([('fd', 'flags')])},
                     {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                     {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                     {'call': 'getpeername',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'setsockopt',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                     {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                     {'call': 'perf_event_open',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev64v2',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
                     {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                     {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                     {'call': 'getsockopt',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'mq_getsetattr',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                     {'call': 'getdents64',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'listen', 'reason': set([('fd', 'flags')])},
                     {'call': 'copy_file_range',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'mq_timedsend',
                      'reason': set([('fd', 'flags')])},
                     {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                     {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                     {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
```

```
                    {'call': 'epoll_wait',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                    {'call': 'preadv64v2',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'readv', 'reason': set([('fd', 'flags')])},
                    {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                    {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                    {'call': 'write', 'reason': set([('fd', 'flags')])},
                    {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                    {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                    {'call': 'bind', 'reason': set([('fd', 'flags')])},
                    {'call': 'flistxattr',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'sendfile64',
                     'reason': set([('fd', 'flags')])}],
 'io_cancel': [{'call': 'io_getevents',
                'reason': set([('kioctx', 'user_id'),
                               ('kioctx_table', 'nr')])},
               {'call': 'io_submit', 'reason': set([('kioctx', 'user_id')])},
               {'call': 'io_setup',
                'reason': set([('kioctx', 'user_id'),
                               ('kioctx_table', 'nr')])},
               {'call': 'io_destroy',
                'reason': set([('kioctx', 'user_id'),
                               ('kioctx_table', 'nr')])}],
 'io_destroy': [{'call': 'io_getevents',
                 'reason': set([('kioctx', 'max_reqs'),
                                ('kioctx', 'mmap_base'),
                                ('kioctx', 'mmap_size'),
                                ('kioctx', 'user_id'),
                                ('kioctx_table', 'nr')])},
                {'call': 'io_submit',
                 'reason': set([('kioctx', 'max_reqs'),
                                ('kioctx', 'mmap_base'),
                                ('kioctx', 'mmap_size'),
                                ('kioctx', 'user_id')])},
                {'call': 'io_setup',
                 'reason': set([('kioctx', 'max_reqs'),
                                ('kioctx', 'mmap_base'),
                                ('kioctx', 'mmap_size'),
                                ('kioctx', 'user_id'),
                                ('kioctx_table', 'nr')])},
                {'call': 'io_cancel',
                 'reason': set([('kioctx', 'max_reqs'),
                                ('kioctx', 'mmap_base'),
                                ('kioctx', 'mmap_size'),
                                ('kioctx', 'user_id'),
                                ('kioctx_table', 'nr')])}],
 'io_getevents': [{'call': 'keyctl',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'rt_sigtimedwait',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'msgrcv',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'kill',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_getaffinity',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setparam',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'ioprio_set',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getppid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'mq_timedreceive',
```



```
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'capget',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setaffinity',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'signal',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'semtimedop',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'umount',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_rr_get_interval',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'rt_sigprocmask',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setsid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sigaltstack',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setattr',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'migrate_pages',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getitimer',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setpgid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getsid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'prlimit64',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'perf_event_open',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'rt_sigaction',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getpgid',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getpriority',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sigaction',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setns',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'fork',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'io_submit',
                   'reason': set([('kioctx', 'user_id')])},
                  {'call': 'get_robust_list',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'mq_timedsend',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_getscheduler',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'ptrace',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_getattr',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'getrusage',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'sched_setscheduler',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'setitimer',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'ioprio_get',
                   'reason': set([('task_struct', 'timer_slack_ns')])},
                  {'call': 'vfork',
```

```
                  'reason': set([('task_struct', 'timer_slack_ns')])},
                 {'call': 'io_setup',
                  'reason': set([('kioctx', 'user_id'),
                                 ('kioctx_table', 'nr')])},
                 {'call': 'io_destroy',
                  'reason': set([('kioctx', 'user_id'),
                                 ('kioctx_table', 'nr')])},
                 {'call': 'prctl',
                  'reason': set([('task_struct', 'timer_slack_ns')])},
                 {'call': 'move_pages',
                  'reason': set([('task_struct', 'timer_slack_ns')])},
                 {'call': 'setpriority',
                  'reason': set([('task_struct', 'timer_slack_ns')])},
                 {'call': 'clone',
                  'reason': set([('task_struct', 'timer_slack_ns')])},
                 {'call': 'sched_getparam',
                  'reason': set([('task_struct', 'timer_slack_ns')])},
                 {'call': 'io_cancel',
                  'reason': set([('kioctx', 'user_id'),
                                 ('kioctx_table', 'nr')])}],
 'io_setup': [{'call': 'io_getevents',
               'reason': set([('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_submit',
               'reason': set([('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_destroy',
               'reason': set([('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])},
              {'call': 'io_cancel',
               'reason': set([('kioctx', 'max_reqs'),
                              ('kioctx', 'mmap_base'),
                              ('kioctx', 'mmap_size'),
                              ('kioctx', 'req_batch')])}],
 'io_submit': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setparam',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioprio_set',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedreceive',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'semtimedop',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_rr_get_interval',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigprocmask',
```



```
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sigaltstack',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setattr',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'migrate_pages',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getitimer',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'prlimit64',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'perf_event_open',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigaction',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getpriority',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sigaction',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'get_robust_list',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedsend',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getscheduler',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getattr',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getrusage',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setscheduler',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setitimer',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioprio_get',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'move_pages',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'setpriority',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getparam',
                'reason': set([('mm_segment_t', 'seg')])}],
 'ioctl': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
           {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
           {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
           {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
           {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
           {'call': 'readahead', 'reason': set([('fd', 'flags')])},
           {'call': 'getdents', 'reason': set([('fd', 'flags')])},
           {'call': 'writev', 'reason': set([('fd', 'flags')])},
           {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
           {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
           {'call': 'pread64', 'reason': set([('fd', 'flags')])},
           {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
           {'call': 'read', 'reason': set([('fd', 'flags')])},
           {'call': 'fchown', 'reason': set([('fd', 'flags')])},
           {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
            {'call': 'flock', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'ioperm': [{'call': 'keyctl',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'iopl',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'msgrcv',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'kill',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_getaffinity',
```



```
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_setparam',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'ioprio_set',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'getppid',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'mq_timedreceive',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'capget',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_setaffinity',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'signal',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'semtimedop',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'umount',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'setsid',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sigaltstack',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_setattr',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'migrate_pages',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'getitimer',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'setpgid',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'getsid',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'prlimit64',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'perf_event_open',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'rt_sigaction',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'getpgid',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'getpriority',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sigaction',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'setns',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'fork',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'get_robust_list',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'mq_timedsend',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_getscheduler',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'ptrace',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_getattr',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'getrusage',
             'reason': set([('thread_struct', 'io_bitmap_ptr')])},
            {'call': 'sched_setscheduler',
```

```
3030              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3031             {'call': 'setitimer',
3032              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3033             {'call': 'ioprio_get',
3034              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3035             {'call': 'vfork',
3036              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3037             {'call': 'prctl',
3038              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3039             {'call': 'move_pages',
3040              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3041             {'call': 'setpriority',
3042              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3043             {'call': 'clone',
3044              'reason': set([('thread_struct', 'io_bitmap_ptr')])},
3045             {'call': 'sched_getparam',
3046              'reason': set([('thread_struct', 'io_bitmap_ptr')])}],
3047  'keyctl': [{'call': 'rt_sigtimedwait',
3048              'reason': set([('task_struct', 'pid')])},
3049             {'call': 'msgrcv', 'reason': set([('task_struct', 'pid')])},
3050             {'call': 'kill', 'reason': set([('task_struct', 'pid')])},
3051             {'call': 'sched_getaffinity',
3052              'reason': set([('task_struct', 'pid')])},
3053             {'call': 'sched_setparam',
3054              'reason': set([('task_struct', 'pid')])},
3055             {'call': 'ioprio_set', 'reason': set([('task_struct', 'pid')])},
3056             {'call': 'getppid', 'reason': set([('task_struct', 'pid')])},
3057             {'call': 'mq_timedreceive',
3058              'reason': set([('task_struct', 'pid')])},
3059             {'call': 'capget', 'reason': set([('task_struct', 'pid')])},
3060             {'call': 'sched_setaffinity',
3061              'reason': set([('task_struct', 'pid')])},
3062             {'call': 'signal', 'reason': set([('task_struct', 'pid')])},
3063             {'call': 'semtimedop', 'reason': set([('task_struct', 'pid')])},
3064             {'call': 'umount', 'reason': set([('task_struct', 'pid')])},
3065             {'call': 'sched_rr_get_interval',
3066              'reason': set([('task_struct', 'pid')])},
3067             {'call': 'rt_sigprocmask',
3068              'reason': set([('task_struct', 'pid')])},
3069             {'call': 'setsid', 'reason': set([('task_struct', 'pid')])},
3070             {'call': 'sigaltstack', 'reason': set([('task_struct', 'pid')])},
3071             {'call': 'sched_setattr',
3072              'reason': set([('task_struct', 'pid')])},
3073             {'call': 'migrate_pages',
3074              'reason': set([('task_struct', 'pid')])},
3075             {'call': 'getitimer', 'reason': set([('task_struct', 'pid')])},
3076             {'call': 'setpgid', 'reason': set([('task_struct', 'pid')])},
3077             {'call': 'getsid', 'reason': set([('task_struct', 'pid')])},
3078             {'call': 'prlimit64', 'reason': set([('task_struct', 'pid')])},
3079             {'call': 'perf_event_open',
3080              'reason': set([('task_struct', 'pid')])},
3081             {'call': 'rt_sigaction', 'reason': set([('task_struct', 'pid')])},
3082             {'call': 'request_key',
3083              'reason': set([('key', 'description'),
3084                             ('key', 'perm'),
3085                             ('key', 'quotalen'),
3086                             ('key', 'serial'),
3087                             ('key_type', 'name'),
3088                             ('key_type', 'read')])},
3089             {'call': 'getpgid', 'reason': set([('task_struct', 'pid')])},
3090             {'call': 'getpriority', 'reason': set([('task_struct', 'pid')])},
3091             {'call': 'sigaction', 'reason': set([('task_struct', 'pid')])},
3092             {'call': 'setns', 'reason': set([('task_struct', 'pid')])},
3093             {'call': 'fork', 'reason': set([('task_struct', 'pid')])},
3094             {'call': 'get_robust_list',
3095              'reason': set([('task_struct', 'pid')])},
```



```
3096             {'call': 'mq_timedsend', 'reason': set([('task_struct', 'pid')])},
3097             {'call': 'sched_getscheduler',
3098              'reason': set([('task_struct', 'pid')])},
3099             {'call': 'ptrace', 'reason': set([('task_struct', 'pid')])},
3100             {'call': 'sched_getattr',
3101              'reason': set([('task_struct', 'pid')])},
3102             {'call': 'getrusage', 'reason': set([('task_struct', 'pid')])},
3103             {'call': 'sched_setscheduler',
3104              'reason': set([('task_struct', 'pid')])},
3105             {'call': 'setitimer', 'reason': set([('task_struct', 'pid')])},
3106             {'call': 'ioprio_get', 'reason': set([('task_struct', 'pid')])},
3107             {'call': 'vfork', 'reason': set([('task_struct', 'pid')])},
3108             {'call': 'prctl', 'reason': set([('task_struct', 'pid')])},
3109             {'call': 'move_pages', 'reason': set([('task_struct', 'pid')])},
3110             {'call': 'setpriority', 'reason': set([('task_struct', 'pid')])},
3111             {'call': 'clone', 'reason': set([('task_struct', 'pid')])},
3112             {'call': 'sched_getparam',
3113              'reason': set([('task_struct', 'pid')])}],
3114  'llseek': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
3115             {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
3116             {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
3117             {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
3118             {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
3119             {'call': 'readahead', 'reason': set([('fd', 'flags')])},
3120             {'call': 'getdents', 'reason': set([('fd', 'flags')])},
3121             {'call': 'writev', 'reason': set([('fd', 'flags')])},
3122             {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
3123             {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
3124             {'call': 'pread64', 'reason': set([('fd', 'flags')])},
3125             {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
3126             {'call': 'read', 'reason': set([('fd', 'flags')])},
3127             {'call': 'fchown', 'reason': set([('fd', 'flags')])},
3128             {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
3129             {'call': 'utime', 'reason': set([('fd', 'flags')])},
3130             {'call': 'fsync', 'reason': set([('fd', 'flags')])},
3131             {'call': 'bpf', 'reason': set([('fd', 'flags')])},
3132             {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
3133             {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
3134             {'call': 'sendto', 'reason': set([('fd', 'flags')])},
3135             {'call': 'tee', 'reason': set([('fd', 'flags')])},
3136             {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
3137             {'call': 'lseek', 'reason': set([('fd', 'flags')])},
3138             {'call': 'connect', 'reason': set([('fd', 'flags')])},
3139             {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
3140             {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
3141             {'call': 'flock', 'reason': set([('fd', 'flags')])},
3142             {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
3143             {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
3144             {'call': 'accept4', 'reason': set([('fd', 'flags')])},
3145             {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
3146             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
3147             {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
3148             {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
3149             {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
3150             {'call': 'splice', 'reason': set([('fd', 'flags')])},
3151             {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
3152             {'call': 'preadv', 'reason': set([('fd', 'flags')])},
3153             {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
3154             {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
3155             {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
3156             {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
3157             {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
3158             {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
3159             {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
3160             {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
3161             {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
```

```
3162             {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
3163             {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
3164             {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
3165             {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
3166             {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
3167             {'call': 'listen', 'reason': set([('fd', 'flags')])},
3168             {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
3169             {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
3170             {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
3171             {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
3172             {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
3173             {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
3174             {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
3175             {'call': 'readv', 'reason': set([('fd', 'flags')])},
3176             {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
3177             {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
3178             {'call': 'write', 'reason': set([('fd', 'flags')])},
3179             {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
3180             {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
3181             {'call': 'bind', 'reason': set([('fd', 'flags')])},
3182             {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
3183             {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
3184  'lseek': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
3185            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
3186            {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
3187            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
3188            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
3189            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
3190            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
3191            {'call': 'writev', 'reason': set([('fd', 'flags')])},
3192            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
3193            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
3194            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
3195            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
3196            {'call': 'read', 'reason': set([('fd', 'flags')])},
3197            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
3198            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
3199            {'call': 'utime', 'reason': set([('fd', 'flags')])},
3200            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
3201            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
3202            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
3203            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
3204            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
3205            {'call': 'tee', 'reason': set([('fd', 'flags')])},
3206            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
3207            {'call': 'connect', 'reason': set([('fd', 'flags')])},
3208            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
3209            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
3210            {'call': 'flock', 'reason': set([('fd', 'flags')])},
3211            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
3212            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
3213            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
3214            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
3215            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
3216            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
3217            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
3218            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
3219            {'call': 'splice', 'reason': set([('fd', 'flags')])},
3220            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
3221            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
3222            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
3223            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
3224            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
3225            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
3226            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
3227            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
```



```
3228            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
3229            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
3230            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
3231            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
3232            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
3233            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
3234            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
3235            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
3236            {'call': 'listen', 'reason': set([('fd', 'flags')])},
3237            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
3238            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
3239            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
3240            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
3241            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
3242            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
3243            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
3244            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
3245            {'call': 'readv', 'reason': set([('fd', 'flags')])},
3246            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
3247            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
3248            {'call': 'write', 'reason': set([('fd', 'flags')])},
3249            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
3250            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
3251            {'call': 'bind', 'reason': set([('fd', 'flags')])},
3252            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
3253            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
3254  'lstat': [{'call': 'stat',
3255             'reason': set([('__old_kernel_stat', 'st_ino'),
3256                            ('__old_kernel_stat', 'st_nlink')])},
3257            {'call': 'fstat',
3258             'reason': set([('__old_kernel_stat', 'st_ino'),
3259                            ('__old_kernel_stat', 'st_nlink')])}],
3260  'madvise': [{'call': 'remap_file_pages',
3261               'reason': set([('vm_area_struct', 'vm_end'),
3262                              ('vm_area_struct', 'vm_start')])},
3263              {'call': 'shmdt',
3264               'reason': set([('vm_area_struct', 'vm_end'),
3265                              ('vm_area_struct', 'vm_start')])},
3266              {'call': 'brk',
3267               'reason': set([('vm_area_struct', 'vm_end'),
3268                              ('vm_area_struct', 'vm_start')])},
3269              {'call': 'get_mempolicy',
3270               'reason': set([('vm_area_struct', 'vm_end'),
3271                              ('vm_area_struct', 'vm_start')])},
3272              {'call': 'munlockall',
3273               'reason': set([('vm_area_struct', 'vm_end'),
3274                              ('vm_area_struct', 'vm_start')])},
3275              {'call': 'pkey_mprotect',
3276               'reason': set([('vm_area_struct', 'vm_end'),
3277                              ('vm_area_struct', 'vm_start')])},
3278              {'call': 'mprotect',
3279               'reason': set([('vm_area_struct', 'vm_end'),
3280                              ('vm_area_struct', 'vm_start')])},
3281              {'call': 'mremap',
3282               'reason': set([('vm_area_struct', 'vm_end'),
3283                              ('vm_area_struct', 'vm_start')])},
3284              {'call': 'prctl',
3285               'reason': set([('vm_area_struct', 'vm_end'),
3286                              ('vm_area_struct', 'vm_start')])},
3287              {'call': 'munlock',
3288               'reason': set([('vm_area_struct', 'vm_end'),
3289                              ('vm_area_struct', 'vm_start')])},
3290              {'call': 'mincore',
3291               'reason': set([('vm_area_struct', 'vm_end'),
3292                              ('vm_area_struct', 'vm_start')])},
3293              {'call': 'msync',
```

```
3294                'reason': set([('vm_area_struct', 'vm_end'),
3295                               ('vm_area_struct', 'vm_start')])},
3296               {'call': 'mlockall',
3297                'reason': set([('vm_area_struct', 'vm_end'),
3298                               ('vm_area_struct', 'vm_start')])}],
3299  'migrate_pages': [{'call': 'keyctl',
3300                     'reason': set([('mm_segment_t', 'seg')])},
3301                    {'call': 'rt_sigtimedwait',
3302                     'reason': set([('mm_segment_t', 'seg')])},
3303                    {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
3304                    {'call': 'msgrcv',
3305                     'reason': set([('mm_segment_t', 'seg')])},
3306                    {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
3307                    {'call': 'sched_getaffinity',
3308                     'reason': set([('mm_segment_t', 'seg')])},
3309                    {'call': 'sched_setparam',
3310                     'reason': set([('mm_segment_t', 'seg')])},
3311                    {'call': 'ioprio_set',
3312                     'reason': set([('mm_segment_t', 'seg')])},
3313                    {'call': 'getppid',
3314                     'reason': set([('mm_segment_t', 'seg')])},
3315                    {'call': 'ioperm',
3316                     'reason': set([('mm_segment_t', 'seg')])},
3317                    {'call': 'mq_timedreceive',
3318                     'reason': set([('mm_segment_t', 'seg')])},
3319                    {'call': 'capget',
3320                     'reason': set([('mm_segment_t', 'seg')])},
3321                    {'call': 'sched_setaffinity',
3322                     'reason': set([('mm_segment_t', 'seg')])},
3323                    {'call': 'signal',
3324                     'reason': set([('mm_segment_t', 'seg')])},
3325                    {'call': 'semtimedop',
3326                     'reason': set([('mm_segment_t', 'seg')])},
3327                    {'call': 'umount',
3328                     'reason': set([('mm_segment_t', 'seg')])},
3329                    {'call': 'sched_rr_get_interval',
3330                     'reason': set([('mm_segment_t', 'seg')])},
3331                    {'call': 'rt_sigprocmask',
3332                     'reason': set([('mm_segment_t', 'seg')])},
3333                    {'call': 'setsid',
3334                     'reason': set([('mm_segment_t', 'seg')])},
3335                    {'call': 'sigaltstack',
3336                     'reason': set([('mm_segment_t', 'seg')])},
3337                    {'call': 'sched_setattr',
3338                     'reason': set([('mm_segment_t', 'seg')])},
3339                    {'call': 'getitimer',
3340                     'reason': set([('mm_segment_t', 'seg')])},
3341                    {'call': 'setpgid',
3342                     'reason': set([('mm_segment_t', 'seg')])},
3343                    {'call': 'getsid',
3344                     'reason': set([('mm_segment_t', 'seg')])},
3345                    {'call': 'prlimit64',
3346                     'reason': set([('mm_segment_t', 'seg')])},
3347                    {'call': 'perf_event_open',
3348                     'reason': set([('mm_segment_t', 'seg')])},
3349                    {'call': 'rt_sigaction',
3350                     'reason': set([('mm_segment_t', 'seg')])},
3351                    {'call': 'getpgid',
3352                     'reason': set([('mm_segment_t', 'seg')])},
3353                    {'call': 'getpriority',
3354                     'reason': set([('mm_segment_t', 'seg')])},
3355                    {'call': 'sigaction',
3356                     'reason': set([('mm_segment_t', 'seg')])},
3357                    {'call': 'setns',
3358                     'reason': set([('mm_segment_t', 'seg')])},
3359                    {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
```



```
3360                    {'call': 'get_robust_list',
3361                     'reason': set([('mm_segment_t', 'seg')])},
3362                    {'call': 'mq_timedsend',
3363                     'reason': set([('mm_segment_t', 'seg')])},
3364                    {'call': 'sched_getscheduler',
3365                     'reason': set([('mm_segment_t', 'seg')])},
3366                    {'call': 'ptrace',
3367                     'reason': set([('mm_segment_t', 'seg')])},
3368                    {'call': 'sched_getattr',
3369                     'reason': set([('mm_segment_t', 'seg')])},
3370                    {'call': 'getrusage',
3371                     'reason': set([('mm_segment_t', 'seg')])},
3372                    {'call': 'sched_setscheduler',
3373                     'reason': set([('mm_segment_t', 'seg')])},
3374                    {'call': 'setitimer',
3375                     'reason': set([('mm_segment_t', 'seg')])},
3376                    {'call': 'ioprio_get',
3377                     'reason': set([('mm_segment_t', 'seg')])},
3378                    {'call': 'vfork',
3379                     'reason': set([('mm_segment_t', 'seg')])},
3380                    {'call': 'prctl',
3381                     'reason': set([('mm_segment_t', 'seg')])},
3382                    {'call': 'move_pages',
3383                     'reason': set([('mm_segment_t', 'seg')])},
3384                    {'call': 'setpriority',
3385                     'reason': set([('mm_segment_t', 'seg')])},
3386                    {'call': 'clone',
3387                     'reason': set([('mm_segment_t', 'seg')])},
3388                    {'call': 'sched_getparam',
3389                     'reason': set([('mm_segment_t', 'seg')])}],
3390  'mincore': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
3391              {'call': 'rt_sigtimedwait',
3392               'reason': set([('mm_segment_t', 'seg')])},
3393              {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
3394              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
3395              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
3396              {'call': 'sched_getaffinity',
3397               'reason': set([('mm_segment_t', 'seg')])},
3398              {'call': 'sched_setparam',
3399               'reason': set([('mm_segment_t', 'seg')])},
3400              {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
3401              {'call': 'remap_file_pages',
3402               'reason': set([('vm_area_struct', 'vm_start')])},
3403              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
3404              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
3405              {'call': 'mq_timedreceive',
3406               'reason': set([('mm_segment_t', 'seg')])},
3407              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
3408              {'call': 'sched_setaffinity',
3409               'reason': set([('mm_segment_t', 'seg')])},
3410              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
3411              {'call': 'semtimedop', 'reason': set([('mm_segment_t', 'seg')])},
3412              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
3413              {'call': 'sched_rr_get_interval',
3414               'reason': set([('mm_segment_t', 'seg')])},
3415              {'call': 'rt_sigprocmask',
3416               'reason': set([('mm_segment_t', 'seg')])},
3417              {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
3418              {'call': 'sigaltstack',
3419               'reason': set([('mm_segment_t', 'seg')])},
3420              {'call': 'sched_setattr',
3421               'reason': set([('mm_segment_t', 'seg')])},
3422              {'call': 'migrate_pages',
3423               'reason': set([('mm_segment_t', 'seg')])},
3424              {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
3425              {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
```

```
3426             {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
3427             {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
3428             {'call': 'perf_event_open',
3429              'reason': set([('mm_segment_t', 'seg')])},
3430             {'call': 'shmdt',
3431              'reason': set([('vm_area_struct', 'vm_start')])},
3432             {'call': 'rt_sigaction',
3433              'reason': set([('mm_segment_t', 'seg')])},
3434             {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
3435             {'call': 'brk', 'reason': set([('vm_area_struct', 'vm_start')])},
3436             {'call': 'getpriority',
3437              'reason': set([('mm_segment_t', 'seg')])},
3438             {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
3439             {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
3440             {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
3441             {'call': 'get_mempolicy',
3442              'reason': set([('vm_area_struct', 'vm_start')])},
3443             {'call': 'get_robust_list',
3444              'reason': set([('mm_segment_t', 'seg')])},
3445             {'call': 'mq_timedsend',
3446              'reason': set([('mm_segment_t', 'seg')])},
3447             {'call': 'sched_getscheduler',
3448              'reason': set([('mm_segment_t', 'seg')])},
3449             {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
3450             {'call': 'munlockall',
3451              'reason': set([('vm_area_struct', 'vm_start')])},
3452             {'call': 'pkey_mprotect',
3453              'reason': set([('vm_area_struct', 'vm_start')])},
3454             {'call': 'madvise',
3455              'reason': set([('vm_area_struct', 'vm_start')])},
3456             {'call': 'sched_getattr',
3457              'reason': set([('mm_segment_t', 'seg')])},
3458             {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
3459             {'call': 'sched_setscheduler',
3460              'reason': set([('mm_segment_t', 'seg')])},
3461             {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
3462             {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
3463             {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
3464             {'call': 'mprotect',
3465              'reason': set([('vm_area_struct', 'vm_start')])},
3466             {'call': 'mremap',
3467              'reason': set([('vm_area_struct', 'vm_start')])},
3468             {'call': 'prctl',
3469              'reason': set([('mm_segment_t', 'seg'),
3470                             ('vm_area_struct', 'vm_start')])},
3471             {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
3472             {'call': 'munlock',
3473              'reason': set([('vm_area_struct', 'vm_start')])},
3474             {'call': 'setpriority',
3475              'reason': set([('mm_segment_t', 'seg')])},
3476             {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
3477             {'call': 'msync',
3478              'reason': set([('vm_area_struct', 'vm_start')])},
3479             {'call': 'sched_getparam',
3480              'reason': set([('mm_segment_t', 'seg')])},
3481             {'call': 'mlockall',
3482              'reason': set([('vm_area_struct', 'vm_start')])}],
3483  'mkdirat': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
3484              {'call': 'fadvise64_64',
3485               'reason': set([('super_block', 's_flags')])},
3486              {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
3487              {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
3488              {'call': 'quotactl',
3489               'reason': set([('super_block', 's_flags')])},
3490              {'call': 'swapon', 'reason': set([('super_block', 's_flags')])}],
3491  'mknodat': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
```



```
3492              {'call': 'fadvise64_64',
3493               'reason': set([('super_block', 's_flags')])},
3494              {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
3495              {'call': 'umount', 'reason': set([('super_block', 's_flags')])},
3496              {'call': 'quotactl',
3497               'reason': set([('super_block', 's_flags')])},
3498              {'call': 'swapon', 'reason': set([('super_block', 's_flags')])}],
3499  'mlockall': [{'call': 'keyctl',
3500                'reason': set([('task_struct', 'personality')])},
3501               {'call': 'rt_sigtimedwait',
3502                'reason': set([('task_struct', 'personality')])},
3503               {'call': 'msgrcv',
3504                'reason': set([('task_struct', 'personality')])},
3505               {'call': 'kill',
3506                'reason': set([('task_struct', 'personality')])},
3507               {'call': 'swapoff', 'reason': set([('mm_struct', 'total_vm')])},
3508               {'call': 'sched_getaffinity',
3509                'reason': set([('task_struct', 'personality')])},
3510               {'call': 'sched_setparam',
3511                'reason': set([('task_struct', 'personality')])},
3512               {'call': 'ioprio_set',
3513                'reason': set([('task_struct', 'personality')])},
3514               {'call': 'personality',
3515                'reason': set([('task_struct', 'personality')])},
3516               {'call': 'remap_file_pages',
3517                'reason': set([('mm_struct', 'total_vm'),
3518                               ('vm_area_struct', 'vm_end'),
3519                               ('vm_area_struct', 'vm_start')])},
3520               {'call': 'io_getevents',
3521                'reason': set([('mm_struct', 'total_vm')])},
3522               {'call': 'getppid',
3523                'reason': set([('task_struct', 'personality')])},
3524               {'call': 'mq_timedreceive',
3525                'reason': set([('task_struct', 'personality')])},
3526               {'call': 'capget',
3527                'reason': set([('task_struct', 'personality')])},
3528               {'call': 'sched_setaffinity',
3529                'reason': set([('task_struct', 'personality')])},
3530               {'call': 'signal',
3531                'reason': set([('task_struct', 'personality')])},
3532               {'call': 'semtimedop',
3533                'reason': set([('task_struct', 'personality')])},
3534               {'call': 'umount',
3535                'reason': set([('task_struct', 'personality')])},
3536               {'call': 'sched_rr_get_interval',
3537                'reason': set([('task_struct', 'personality')])},
3538               {'call': 'rt_sigprocmask',
3539                'reason': set([('task_struct', 'personality')])},
3540               {'call': 'setsid',
3541                'reason': set([('task_struct', 'personality')])},
3542               {'call': 'sigaltstack',
3543                'reason': set([('task_struct', 'personality')])},
3544               {'call': 'sched_setattr',
3545                'reason': set([('task_struct', 'personality')])},
3546               {'call': 'migrate_pages',
3547                'reason': set([('mm_struct', 'total_vm'),
3548                               ('task_struct', 'personality')])},
3549               {'call': 'getitimer',
3550                'reason': set([('task_struct', 'personality')])},
3551               {'call': 'setpgid',
3552                'reason': set([('task_struct', 'personality')])},
3553               {'call': 'getsid',
3554                'reason': set([('task_struct', 'personality')])},
3555               {'call': 'prlimit64',
3556                'reason': set([('task_struct', 'personality')])},
3557               {'call': 'perf_event_open',
```

```
3558                'reason': set([('task_struct', 'personality')])},
3559               {'call': 'shmdt',
3560                'reason': set([('mm_struct', 'total_vm'),
3561                               ('vm_area_struct', 'vm_end'),
3562                               ('vm_area_struct', 'vm_start')])},
3563               {'call': 'rt_sigaction',
3564                'reason': set([('task_struct', 'personality')])},
3565               {'call': 'getpgid',
3566                'reason': set([('task_struct', 'personality')])},
3567               {'call': 'brk',
3568                'reason': set([('mm_struct', 'total_vm'),
3569                               ('vm_area_struct', 'vm_end'),
3570                               ('vm_area_struct', 'vm_start')])},
3571               {'call': 'getpriority',
3572                'reason': set([('task_struct', 'personality')])},
3573               {'call': 'sigaction',
3574                'reason': set([('task_struct', 'personality')])},
3575               {'call': 'setns',
3576                'reason': set([('task_struct', 'personality')])},
3577               {'call': 'fork',
3578                'reason': set([('task_struct', 'personality')])},
3579               {'call': 'get_mempolicy',
3580                'reason': set([('mm_struct', 'total_vm'),
3581                               ('vm_area_struct', 'vm_end'),
3582                               ('vm_area_struct', 'vm_start')])},
3583               {'call': 'get_robust_list',
3584                'reason': set([('task_struct', 'personality')])},
3585               {'call': 'mq_timedsend',
3586                'reason': set([('task_struct', 'personality')])},
3587               {'call': 'sched_getscheduler',
3588                'reason': set([('task_struct', 'personality')])},
3589               {'call': 'ptrace',
3590                'reason': set([('task_struct', 'personality')])},
3591               {'call': 'munlockall',
3592                'reason': set([('vm_area_struct', 'vm_end'),
3593                               ('vm_area_struct', 'vm_start')])},
3594               {'call': 'pkey_mprotect',
3595                'reason': set([('vm_area_struct', 'vm_end'),
3596                               ('vm_area_struct', 'vm_start')])},
3597               {'call': 'madvise',
3598                'reason': set([('vm_area_struct', 'vm_end'),
3599                               ('vm_area_struct', 'vm_start')])},
3600               {'call': 'sched_getattr',
3601                'reason': set([('task_struct', 'personality')])},
3602               {'call': 'getrusage',
3603                'reason': set([('mm_struct', 'total_vm'),
3604                               ('task_struct', 'personality')])},
3605               {'call': 'sched_setscheduler',
3606                'reason': set([('task_struct', 'personality')])},
3607               {'call': 'setitimer',
3608                'reason': set([('task_struct', 'personality')])},
3609               {'call': 'ioprio_get',
3610                'reason': set([('task_struct', 'personality')])},
3611               {'call': 'vfork',
3612                'reason': set([('task_struct', 'personality')])},
3613               {'call': 'io_setup',
3614                'reason': set([('mm_struct', 'total_vm')])},
3615               {'call': 'mprotect',
3616                'reason': set([('vm_area_struct', 'vm_end'),
3617                               ('vm_area_struct', 'vm_start')])},
3618               {'call': 'mremap',
3619                'reason': set([('mm_struct', 'total_vm'),
3620                               ('vm_area_struct', 'vm_end'),
3621                               ('vm_area_struct', 'vm_start')])},
3622               {'call': 'io_destroy',
3623                'reason': set([('mm_struct', 'total_vm')])},
```



```
3624               {'call': 'mbind', 'reason': set([('mm_struct', 'total_vm')])},
3625               {'call': 'prctl',
3626                'reason': set([('mm_struct', 'total_vm'),
3627                               ('task_struct', 'personality'),
3628                               ('vm_area_struct', 'vm_end'),
3629                               ('vm_area_struct', 'vm_start')])},
3630               {'call': 'move_pages',
3631                'reason': set([('mm_struct', 'total_vm'),
3632                               ('task_struct', 'personality')])},
3633               {'call': 'modify_ldt',
3634                'reason': set([('mm_struct', 'total_vm')])},
3635               {'call': 'munlock',
3636                'reason': set([('vm_area_struct', 'vm_end'),
3637                               ('vm_area_struct', 'vm_start')])},
3638               {'call': 'setpriority',
3639                'reason': set([('task_struct', 'personality')])},
3640               {'call': 'mincore',
3641                'reason': set([('mm_struct', 'total_vm'),
3642                               ('vm_area_struct', 'vm_end'),
3643                               ('vm_area_struct', 'vm_start')])},
3644               {'call': 'clone',
3645                'reason': set([('task_struct', 'personality')])},
3646               {'call': 'msync',
3647                'reason': set([('mm_struct', 'total_vm'),
3648                               ('vm_area_struct', 'vm_end'),
3649                               ('vm_area_struct', 'vm_start')])},
3650               {'call': 'sched_getparam',
3651                'reason': set([('task_struct', 'personality')])},
3652               {'call': 'io_cancel',
3653                'reason': set([('mm_struct', 'total_vm')])}],
3654  'modify_ldt': [{'call': 'get_thread_area',
3655                  'reason': set([('user_desc', 'base_addr'),
3656                                 ('user_desc', 'contents'),
3657                                 ('user_desc', 'entry_number'),
3658                                 ('user_desc', 'limit'),
3659                                 ('user_desc', 'limit_in_pages'),
3660                                 ('user_desc', 'read_exec_only'),
3661                                 ('user_desc', 'seg_32bit'),
3662                                 ('user_desc', 'seg_not_present'),
3663                                 ('user_desc', 'useable')])},
3664                 {'call': 'set_thread_area',
3665                  'reason': set([('user_desc', 'base_addr'),
3666                                 ('user_desc', 'contents'),
3667                                 ('user_desc', 'entry_number'),
3668                                 ('user_desc', 'limit'),
3669                                 ('user_desc', 'limit_in_pages'),
3670                                 ('user_desc', 'read_exec_only'),
3671                                 ('user_desc', 'seg_32bit'),
3672                                 ('user_desc', 'seg_not_present'),
3673                                 ('user_desc', 'useable')])}],
3674  'mount': [{'call': 'keyctl',
3675             'reason': set([('task_struct', 'personality')])},
3676            {'call': 'rt_sigtimedwait',
3677             'reason': set([('task_struct', 'personality')])},
3678            {'call': 'msgrcv',
3679             'reason': set([('task_struct', 'personality')])},
3680            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
3681            {'call': 'sched_getaffinity',
3682             'reason': set([('task_struct', 'personality')])},
3683            {'call': 'sched_setparam',
3684             'reason': set([('task_struct', 'personality')])},
3685            {'call': 'ioprio_set',
3686             'reason': set([('task_struct', 'personality')])},
3687            {'call': 'personality',
3688             'reason': set([('task_struct', 'personality')])},
3689            {'call': 'getppid',
```

```
3690             'reason': set([('task_struct', 'personality')])},
3691            {'call': 'mq_timedreceive',
3692             'reason': set([('task_struct', 'personality')])},
3693            {'call': 'capget',
3694             'reason': set([('task_struct', 'personality')])},
3695            {'call': 'sched_setaffinity',
3696             'reason': set([('task_struct', 'personality')])},
3697            {'call': 'signal',
3698             'reason': set([('task_struct', 'personality')])},
3699            {'call': 'semtimedop',
3700             'reason': set([('task_struct', 'personality')])},
3701            {'call': 'umount',
3702             'reason': set([('task_struct', 'personality')])},
3703            {'call': 'sched_rr_get_interval',
3704             'reason': set([('task_struct', 'personality')])},
3705            {'call': 'rt_sigprocmask',
3706             'reason': set([('task_struct', 'personality')])},
3707            {'call': 'setsid',
3708             'reason': set([('task_struct', 'personality')])},
3709            {'call': 'sigaltstack',
3710             'reason': set([('task_struct', 'personality')])},
3711            {'call': 'sched_setattr',
3712             'reason': set([('task_struct', 'personality')])},
3713            {'call': 'migrate_pages',
3714             'reason': set([('task_struct', 'personality')])},
3715            {'call': 'getitimer',
3716             'reason': set([('task_struct', 'personality')])},
3717            {'call': 'setpgid',
3718             'reason': set([('task_struct', 'personality')])},
3719            {'call': 'getsid',
3720             'reason': set([('task_struct', 'personality')])},
3721            {'call': 'prlimit64',
3722             'reason': set([('task_struct', 'personality')])},
3723            {'call': 'perf_event_open',
3724             'reason': set([('task_struct', 'personality')])},
3725            {'call': 'rt_sigaction',
3726             'reason': set([('task_struct', 'personality')])},
3727            {'call': 'getpgid',
3728             'reason': set([('task_struct', 'personality')])},
3729            {'call': 'getpriority',
3730             'reason': set([('task_struct', 'personality')])},
3731            {'call': 'sigaction',
3732             'reason': set([('task_struct', 'personality')])},
3733            {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
3734            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
3735            {'call': 'get_robust_list',
3736             'reason': set([('task_struct', 'personality')])},
3737            {'call': 'mq_timedsend',
3738             'reason': set([('task_struct', 'personality')])},
3739            {'call': 'sched_getscheduler',
3740             'reason': set([('task_struct', 'personality')])},
3741            {'call': 'ptrace',
3742             'reason': set([('task_struct', 'personality')])},
3743            {'call': 'sched_getattr',
3744             'reason': set([('task_struct', 'personality')])},
3745            {'call': 'getrusage',
3746             'reason': set([('task_struct', 'personality')])},
3747            {'call': 'sched_setscheduler',
3748             'reason': set([('task_struct', 'personality')])},
3749            {'call': 'setitimer',
3750             'reason': set([('task_struct', 'personality')])},
3751            {'call': 'ioprio_get',
3752             'reason': set([('task_struct', 'personality')])},
3753            {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
3754            {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
3755            {'call': 'move_pages',
```



```
3756             'reason': set([('task_struct', 'personality')])},
3757            {'call': 'setpriority',
3758             'reason': set([('task_struct', 'personality')])},
3759            {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
3760            {'call': 'sched_getparam',
3761             'reason': set([('task_struct', 'personality')])}],
3762  'mprotect': [{'call': 'keyctl',
3763                'reason': set([('task_struct', 'personality')])},
3764               {'call': 'rt_sigtimedwait',
3765                'reason': set([('task_struct', 'personality')])},
3766               {'call': 'msgrcv',
3767                'reason': set([('task_struct', 'personality')])},
3768               {'call': 'kill',
3769                'reason': set([('task_struct', 'personality')])},
3770               {'call': 'sched_getaffinity',
3771                'reason': set([('task_struct', 'personality')])},
3772               {'call': 'sched_setparam',
3773                'reason': set([('task_struct', 'personality')])},
3774               {'call': 'ioprio_set',
3775                'reason': set([('task_struct', 'personality')])},
3776               {'call': 'personality',
3777                'reason': set([('task_struct', 'personality')])},
3778               {'call': 'remap_file_pages',
3779                'reason': set([('vm_area_struct', 'vm_end'),
3780                               ('vm_area_struct', 'vm_flags'),
3781                               ('vm_area_struct', 'vm_start')])},
3782               {'call': 'getppid',
3783                'reason': set([('task_struct', 'personality')])},
3784               {'call': 'mq_timedreceive',
3785                'reason': set([('task_struct', 'personality')])},
3786               {'call': 'capget',
3787                'reason': set([('task_struct', 'personality')])},
3788               {'call': 'sched_setaffinity',
3789                'reason': set([('task_struct', 'personality')])},
3790               {'call': 'signal',
3791                'reason': set([('task_struct', 'personality')])},
3792               {'call': 'semtimedop',
3793                'reason': set([('task_struct', 'personality')])},
3794               {'call': 'umount',
3795                'reason': set([('task_struct', 'personality')])},
3796               {'call': 'sched_rr_get_interval',
3797                'reason': set([('task_struct', 'personality')])},
3798               {'call': 'rt_sigprocmask',
3799                'reason': set([('task_struct', 'personality')])},
3800               {'call': 'setsid',
3801                'reason': set([('task_struct', 'personality')])},
3802               {'call': 'sigaltstack',
3803                'reason': set([('task_struct', 'personality')])},
3804               {'call': 'sched_setattr',
3805                'reason': set([('task_struct', 'personality')])},
3806               {'call': 'migrate_pages',
3807                'reason': set([('task_struct', 'personality')])},
3808               {'call': 'getitimer',
3809                'reason': set([('task_struct', 'personality')])},
3810               {'call': 'setpgid',
3811                'reason': set([('task_struct', 'personality')])},
3812               {'call': 'getsid',
3813                'reason': set([('task_struct', 'personality')])},
3814               {'call': 'prlimit64',
3815                'reason': set([('task_struct', 'personality')])},
3816               {'call': 'perf_event_open',
3817                'reason': set([('task_struct', 'personality')])},
3818               {'call': 'shmdt',
3819                'reason': set([('vm_area_struct', 'vm_end'),
3820                               ('vm_area_struct', 'vm_flags'),
3821                               ('vm_area_struct', 'vm_start')])},
```

```
3822                {'call': 'rt_sigaction',
3823                 'reason': set([('task_struct', 'personality')])},
3824                {'call': 'getpgid',
3825                 'reason': set([('task_struct', 'personality')])},
3826                {'call': 'brk',
3827                 'reason': set([('vm_area_struct', 'vm_end'),
3828                                ('vm_area_struct', 'vm_flags'),
3829                                ('vm_area_struct', 'vm_start')])},
3830                {'call': 'getpriority',
3831                 'reason': set([('task_struct', 'personality')])},
3832                {'call': 'sigaction',
3833                 'reason': set([('task_struct', 'personality')])},
3834                {'call': 'setns',
3835                 'reason': set([('task_struct', 'personality')])},
3836                {'call': 'fork',
3837                 'reason': set([('task_struct', 'personality')])},
3838                {'call': 'get_mempolicy',
3839                 'reason': set([('vm_area_struct', 'vm_end'),
3840                                ('vm_area_struct', 'vm_flags'),
3841                                ('vm_area_struct', 'vm_start')])},
3842                {'call': 'get_robust_list',
3843                 'reason': set([('task_struct', 'personality')])},
3844                {'call': 'mq_timedsend',
3845                 'reason': set([('task_struct', 'personality')])},
3846                {'call': 'sched_getscheduler',
3847                 'reason': set([('task_struct', 'personality')])},
3848                {'call': 'ptrace',
3849                 'reason': set([('task_struct', 'personality')])},
3850                {'call': 'munlockall',
3851                 'reason': set([('vm_area_struct', 'vm_end'),
3852                                ('vm_area_struct', 'vm_flags'),
3853                                ('vm_area_struct', 'vm_start')])},
3854                {'call': 'pkey_mprotect',
3855                 'reason': set([('vm_area_struct', 'vm_end'),
3856                                ('vm_area_struct', 'vm_flags'),
3857                                ('vm_area_struct', 'vm_start')])},
3858                {'call': 'madvise',
3859                 'reason': set([('vm_area_struct', 'vm_end'),
3860                                ('vm_area_struct', 'vm_flags'),
3861                                ('vm_area_struct', 'vm_start')])},
3862                {'call': 'sched_getattr',
3863                 'reason': set([('task_struct', 'personality')])},
3864                {'call': 'getrusage',
3865                 'reason': set([('task_struct', 'personality')])},
3866                {'call': 'sched_setscheduler',
3867                 'reason': set([('task_struct', 'personality')])},
3868                {'call': 'setitimer',
3869                 'reason': set([('task_struct', 'personality')])},
3870                {'call': 'ioprio_get',
3871                 'reason': set([('task_struct', 'personality')])},
3872                {'call': 'vfork',
3873                 'reason': set([('task_struct', 'personality')])},
3874                {'call': 'mremap',
3875                 'reason': set([('vm_area_struct', 'vm_end'),
3876                                ('vm_area_struct', 'vm_flags'),
3877                                ('vm_area_struct', 'vm_start')])},
3878                {'call': 'prctl',
3879                 'reason': set([('task_struct', 'personality'),
3880                                ('vm_area_struct', 'vm_end'),
3881                                ('vm_area_struct', 'vm_flags'),
3882                                ('vm_area_struct', 'vm_start')])},
3883                {'call': 'move_pages',
3884                 'reason': set([('task_struct', 'personality')])},
3885                {'call': 'munlock',
3886                 'reason': set([('vm_area_struct', 'vm_end'),
3887                                ('vm_area_struct', 'vm_flags'),
```



```
3888                                ('vm_area_struct', 'vm_start')])},
3889                {'call': 'setpriority',
3890                 'reason': set([('task_struct', 'personality')])},
3891                {'call': 'mincore',
3892                 'reason': set([('vm_area_struct', 'vm_end'),
3893                                ('vm_area_struct', 'vm_flags'),
3894                                ('vm_area_struct', 'vm_start')])},
3895                {'call': 'clone',
3896                 'reason': set([('task_struct', 'personality')])},
3897                {'call': 'msync',
3898                 'reason': set([('vm_area_struct', 'vm_end'),
3899                                ('vm_area_struct', 'vm_flags'),
3900                                ('vm_area_struct', 'vm_start')])},
3901                {'call': 'sched_getparam',
3902                 'reason': set([('task_struct', 'personality')])},
3903                {'call': 'mlockall',
3904                 'reason': set([('vm_area_struct', 'vm_end'),
3905                                ('vm_area_struct', 'vm_flags'),
3906                                ('vm_area_struct', 'vm_start')])}],
3907  'mq_getsetattr': [{'call': 'mq_timedreceive',
3908                     'reason': set([('mq_attr', 'mq_flags')])},
3909                    {'call': 'mq_timedsend',
3910                     'reason': set([('mq_attr', 'mq_flags')])},
3911                    {'call': 'mq_notify',
3912                     'reason': set([('mq_attr', 'mq_flags')])},
3913                    {'call': 'mq_open',
3914                     'reason': set([('mq_attr', 'mq_flags')])}],
3915  'mq_notify': [{'call': 'rt_sigtimedwait',
3916                 'reason': set([('sigval', 'sival_ptr')])},
3917                {'call': 'mq_timedreceive',
3918                 'reason': set([('sigevent', 'sigev_notify'),
3919                                ('sigevent', 'sigev_signo'),
3920                                ('sigval', 'sival_ptr')])},
3921                {'call': 'timer_create',
3922                 'reason': set([('sigevent', 'sigev_notify'),
3923                                ('sigevent', 'sigev_signo'),
3924                                ('sigval', 'sival_ptr')])},
3925                {'call': 'rt_sigqueueinfo',
3926                 'reason': set([('sigval', 'sival_ptr')])},
3927                {'call': 'tgkill', 'reason': set([('sigval', 'sival_ptr')])},
3928                {'call': 'rt_tgsigqueueinfo',
3929                 'reason': set([('sigval', 'sival_ptr')])},
3930                {'call': 'mq_getsetattr',
3931                 'reason': set([('sigevent', 'sigev_notify'),
3932                                ('sigevent', 'sigev_signo'),
3933                                ('sigval', 'sival_ptr')])},
3934                {'call': 'mq_timedsend',
3935                 'reason': set([('sigevent', 'sigev_notify'),
3936                                ('sigevent', 'sigev_signo'),
3937                                ('sigval', 'sival_ptr')])},
3938                {'call': 'rt_sigreturn',
3939                 'reason': set([('sigval', 'sival_ptr')])},
3940                {'call': 'tkill', 'reason': set([('sigval', 'sival_ptr')])}],
3941  'mq_open': [{'call': 'sysfs', 'reason': set([('filename', 'name')])},
3942              {'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
3943              {'call': 'swapoff', 'reason': set([('filename', 'name')])},
3944              {'call': 'openat', 'reason': set([('filename', 'name')])},
3945              {'call': 'uselib', 'reason': set([('filename', 'name')])},
3946              {'call': 'renameat2', 'reason': set([('filename', 'name')])},
3947              {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
3948              {'call': 'quotactl', 'reason': set([('filename', 'name')])},
3949              {'call': 'acct', 'reason': set([('filename', 'name')])},
3950              {'call': 'open', 'reason': set([('filename', 'name')])},
3951              {'call': 'unlink', 'reason': set([('filename', 'name')])},
3952              {'call': 'rmdir', 'reason': set([('filename', 'name')])},
3953              {'call': 'swapon', 'reason': set([('filename', 'name')])},
```

```
3954             {'call': 'unlinkat', 'reason': set([('filename', 'name')])}],
3955  'mq_timedreceive': [{'call': 'msgrcv', 'reason': set([('msg_msg', 'm_ts')])},
3956                      {'call': 'fadvise64_64',
3957                       'reason': set([('file', 'f_mode')])},
3958                      {'call': 'swapoff',
3959                       'reason': set([('file', 'f_flags'),
3960                                      ('file', 'f_mode')])},
3961                      {'call': 'memfd_create',
3962                       'reason': set([('file', 'f_flags'),
3963                                      ('file', 'f_mode')])},
3964                      {'call': 'remap_file_pages',
3965                       'reason': set([('file', 'f_flags'),
3966                                      ('file', 'f_mode')])},
3967                      {'call': 'dup3',
3968                       'reason': set([('file', 'f_flags'),
3969                                      ('file', 'f_mode')])},
3970                      {'call': 'socketpair',
3971                       'reason': set([('file', 'f_flags'),
3972                                      ('file', 'f_mode')])},
3973                      {'call': 'epoll_create1',
3974                       'reason': set([('file', 'f_flags'),
3975                                      ('file', 'f_mode')])},
3976                      {'call': 'epoll_ctl',
3977                       'reason': set([('file', 'f_flags'),
3978                                      ('file', 'f_mode')])},
3979                      {'call': 'flock',
3980                       'reason': set([('file', 'f_flags'),
3981                                      ('file', 'f_mode')])},
3982                      {'call': 'openat',
3983                       'reason': set([('file', 'f_flags'),
3984                                      ('file', 'f_mode')])},
3985                      {'call': 'uselib',
3986                       'reason': set([('file', 'f_flags'),
3987                                      ('file', 'f_mode')])},
3988                      {'call': 'accept4',
3989                       'reason': set([('file', 'f_flags'),
3990                                      ('file', 'f_mode')])},
3991                      {'call': 'shmat',
3992                       'reason': set([('file', 'f_flags'),
3993                                      ('file', 'f_mode')])},
3994                      {'call': 'socket',
3995                       'reason': set([('file', 'f_flags'),
3996                                      ('file', 'f_mode')])},
3997                      {'call': 'pipe2',
3998                       'reason': set([('file', 'f_flags'),
3999                                      ('file', 'f_mode')])},
4000                      {'call': 'perf_event_open',
4001                       'reason': set([('file', 'f_flags'),
4002                                      ('file', 'f_mode')])},
4003                      {'call': 'shmdt',
4004                       'reason': set([('file', 'f_flags'),
4005                                      ('file', 'f_mode')])},
4006                      {'call': 'acct',
4007                       'reason': set([('file', 'f_flags'),
4008                                      ('file', 'f_mode')])},
4009                      {'call': 'open',
4010                       'reason': set([('file', 'f_flags'),
4011                                      ('file', 'f_mode')])},
4012                      {'call': 'mq_getsetattr',
4013                       'reason': set([('file', 'f_flags'),
4014                                      ('mq_attr', 'mq_curmsgs'),
4015                                      ('mq_attr', 'mq_msgsize')])},
4016                      {'call': 'dup',
4017                       'reason': set([('file', 'f_flags'),
4018                                      ('file', 'f_mode')])},
4019                      {'call': 'setns',
```



```
4020                       'reason': set([('file', 'f_flags'),
4021                                      ('file', 'f_mode')])},
4022                      {'call': 'mq_timedsend',
4023                       'reason': set([('mq_attr', 'mq_curmsgs'),
4024                                      ('mq_attr', 'mq_msgsize'),
4025                                      ('msg_msg', 'm_ts')])},
4026                      {'call': 'shmctl',
4027                       'reason': set([('file', 'f_flags'),
4028                                      ('file', 'f_mode')])},
4029                      {'call': 'swapon',
4030                       'reason': set([('file', 'f_flags'),
4031                                      ('file', 'f_mode')])},
4032                      {'call': 'eventfd2',
4033                       'reason': set([('file', 'f_flags'),
4034                                      ('file', 'f_mode')])},
4035                      {'call': 'mmap_pgoff',
4036                       'reason': set([('file', 'f_flags'),
4037                                      ('file', 'f_mode')])},
4038                      {'call': 'msgsnd', 'reason': set([('msg_msg', 'm_ts')])},
4039                      {'call': 'mq_notify',
4040                       'reason': set([('mq_attr', 'mq_curmsgs'),
4041                                      ('mq_attr', 'mq_msgsize')])},
4042                      {'call': 'mq_open',
4043                       'reason': set([('file', 'f_flags'),
4044                                      ('file', 'f_mode'),
4045                                      ('mq_attr', 'mq_curmsgs'),
4046                                      ('mq_attr', 'mq_msgsize')])},
4047                      {'call': 'msync',
4048                       'reason': set([('file', 'f_flags'),
4049                                      ('file', 'f_mode')])},
4050                      {'call': 'open_by_handle_at',
4051                       'reason': set([('file', 'f_flags'),
4052                                      ('file', 'f_mode')])}],
4053  'mq_timedsend': [{'call': 'fadvise64_64',
4054                    'reason': set([('file', 'f_mode')])},
4055                   {'call': 'swapoff',
4056                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4057                   {'call': 'memfd_create',
4058                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4059                   {'call': 'remap_file_pages',
4060                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4061                   {'call': 'dup3',
4062                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4063                   {'call': 'mq_timedreceive',
4064                    'reason': set([('mq_attr', 'mq_curmsgs'),
4065                                   ('mq_attr', 'mq_maxmsg'),
4066                                   ('mq_attr', 'mq_msgsize')])},
4067                   {'call': 'socketpair',
4068                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4069                   {'call': 'epoll_create1',
4070                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4071                   {'call': 'epoll_ctl',
4072                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4073                   {'call': 'flock',
4074                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4075                   {'call': 'openat',
4076                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4077                   {'call': 'uselib',
4078                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4079                   {'call': 'accept4',
4080                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4081                   {'call': 'shmat',
4082                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4083                   {'call': 'socket',
4084                    'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4085                   {'call': 'pipe2',
```

```
4086                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4087                 {'call': 'perf_event_open',
4088                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4089                 {'call': 'shmdt',
4090                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4091                 {'call': 'acct',
4092                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4093                 {'call': 'open',
4094                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4095                 {'call': 'mq_getsetattr',
4096                  'reason': set([('file', 'f_flags'),
4097                                 ('mq_attr', 'mq_curmsgs'),
4098                                 ('mq_attr', 'mq_maxmsg'),
4099                                 ('mq_attr', 'mq_msgsize')])},
4100                 {'call': 'dup',
4101                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4102                 {'call': 'setns',
4103                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4104                 {'call': 'shmctl',
4105                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4106                 {'call': 'swapon',
4107                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4108                 {'call': 'eventfd2',
4109                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4110                 {'call': 'mmap_pgoff',
4111                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4112                 {'call': 'mq_notify',
4113                  'reason': set([('mq_attr', 'mq_curmsgs'),
4114                                 ('mq_attr', 'mq_maxmsg'),
4115                                 ('mq_attr', 'mq_msgsize')])},
4116                 {'call': 'mq_open',
4117                  'reason': set([('file', 'f_flags'),
4118                                 ('file', 'f_mode'),
4119                                 ('mq_attr', 'mq_curmsgs'),
4120                                 ('mq_attr', 'mq_maxmsg'),
4121                                 ('mq_attr', 'mq_msgsize')])},
4122                 {'call': 'msync',
4123                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
4124                 {'call': 'open_by_handle_at',
4125                  'reason': set([('file', 'f_flags'), ('file', 'f_mode')])}],
4126  'mremap': [{'call': 'keyctl',
4127              'reason': set([('task_struct', 'personality')])},
4128             {'call': 'rt_sigtimedwait',
4129              'reason': set([('task_struct', 'personality')])},
4130             {'call': 'msgrcv',
4131              'reason': set([('task_struct', 'personality')])},
4132             {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
4133             {'call': 'swapoff', 'reason': set([('mm_struct', 'map_count')])},
4134             {'call': 'sched_getaffinity',
4135              'reason': set([('task_struct', 'personality')])},
4136             {'call': 'sched_setparam',
4137              'reason': set([('task_struct', 'personality')])},
4138             {'call': 'ioprio_set',
4139              'reason': set([('task_struct', 'personality')])},
4140             {'call': 'personality',
4141              'reason': set([('task_struct', 'personality')])},
4142             {'call': 'remap_file_pages',
4143              'reason': set([('mm_struct', 'map_count'),
4144                             ('vm_area_struct', 'vm_end'),
4145                             ('vm_area_struct', 'vm_flags'),
4146                             ('vm_area_struct', 'vm_pgoff'),
4147                             ('vm_area_struct', 'vm_start')])},
4148             {'call': 'io_getevents',
4149              'reason': set([('mm_struct', 'map_count')])},
4150             {'call': 'getppid',
4151              'reason': set([('task_struct', 'personality')])},
```



```
4152             {'call': 'mq_timedreceive',
4153              'reason': set([('task_struct', 'personality')])},
4154             {'call': 'capget',
4155              'reason': set([('task_struct', 'personality')])},
4156             {'call': 'sched_setaffinity',
4157              'reason': set([('task_struct', 'personality')])},
4158             {'call': 'signal',
4159              'reason': set([('task_struct', 'personality')])},
4160             {'call': 'semtimedop',
4161              'reason': set([('task_struct', 'personality')])},
4162             {'call': 'umount',
4163              'reason': set([('task_struct', 'personality')])},
4164             {'call': 'sched_rr_get_interval',
4165              'reason': set([('task_struct', 'personality')])},
4166             {'call': 'rt_sigprocmask',
4167              'reason': set([('task_struct', 'personality')])},
4168             {'call': 'setsid',
4169              'reason': set([('task_struct', 'personality')])},
4170             {'call': 'sigaltstack',
4171              'reason': set([('task_struct', 'personality')])},
4172             {'call': 'sched_setattr',
4173              'reason': set([('task_struct', 'personality')])},
4174             {'call': 'migrate_pages',
4175              'reason': set([('mm_struct', 'map_count'),
4176                             ('task_struct', 'personality')])},
4177             {'call': 'getitimer',
4178              'reason': set([('task_struct', 'personality')])},
4179             {'call': 'setpgid',
4180              'reason': set([('task_struct', 'personality')])},
4181             {'call': 'getsid',
4182              'reason': set([('task_struct', 'personality')])},
4183             {'call': 'prlimit64',
4184              'reason': set([('task_struct', 'personality')])},
4185             {'call': 'perf_event_open',
4186              'reason': set([('task_struct', 'personality')])},
4187             {'call': 'shmdt',
4188              'reason': set([('mm_struct', 'map_count'),
4189                             ('vm_area_struct', 'vm_end'),
4190                             ('vm_area_struct', 'vm_flags'),
4191                             ('vm_area_struct', 'vm_pgoff'),
4192                             ('vm_area_struct', 'vm_start')])},
4193             {'call': 'rt_sigaction',
4194              'reason': set([('task_struct', 'personality')])},
4195             {'call': 'getpgid',
4196              'reason': set([('task_struct', 'personality')])},
4197             {'call': 'brk',
4198              'reason': set([('mm_struct', 'map_count'),
4199                             ('vm_area_struct', 'vm_end'),
4200                             ('vm_area_struct', 'vm_flags'),
4201                             ('vm_area_struct', 'vm_pgoff'),
4202                             ('vm_area_struct', 'vm_start')])},
4203             {'call': 'getpriority',
4204              'reason': set([('task_struct', 'personality')])},
4205             {'call': 'sigaction',
4206              'reason': set([('task_struct', 'personality')])},
4207             {'call': 'setns',
4208              'reason': set([('task_struct', 'personality')])},
4209             {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
4210             {'call': 'get_mempolicy',
4211              'reason': set([('mm_struct', 'map_count'),
4212                             ('vm_area_struct', 'vm_end'),
4213                             ('vm_area_struct', 'vm_flags'),
4214                             ('vm_area_struct', 'vm_pgoff'),
4215                             ('vm_area_struct', 'vm_start')])},
4216             {'call': 'get_robust_list',
4217              'reason': set([('task_struct', 'personality')])},
```

```
4218            {'call': 'mq_timedsend',
4219             'reason': set([('task_struct', 'personality')])},
4220            {'call': 'sched_getscheduler',
4221             'reason': set([('task_struct', 'personality')])},
4222            {'call': 'ptrace',
4223             'reason': set([('task_struct', 'personality')])},
4224            {'call': 'munlockall',
4225             'reason': set([('vm_area_struct', 'vm_end'),
4226                            ('vm_area_struct', 'vm_flags'),
4227                            ('vm_area_struct', 'vm_pgoff'),
4228                            ('vm_area_struct', 'vm_start')])},
4229            {'call': 'pkey_mprotect',
4230             'reason': set([('vm_area_struct', 'vm_end'),
4231                            ('vm_area_struct', 'vm_flags'),
4232                            ('vm_area_struct', 'vm_pgoff'),
4233                            ('vm_area_struct', 'vm_start')])},
4234            {'call': 'madvise',
4235             'reason': set([('vm_area_struct', 'vm_end'),
4236                            ('vm_area_struct', 'vm_flags'),
4237                            ('vm_area_struct', 'vm_pgoff'),
4238                            ('vm_area_struct', 'vm_start')])},
4239            {'call': 'sched_getattr',
4240             'reason': set([('task_struct', 'personality')])},
4241            {'call': 'getrusage',
4242             'reason': set([('mm_struct', 'map_count'),
4243                            ('task_struct', 'personality')])},
4244            {'call': 'sched_setscheduler',
4245             'reason': set([('task_struct', 'personality')])},
4246            {'call': 'setitimer',
4247             'reason': set([('task_struct', 'personality')])},
4248            {'call': 'ioprio_get',
4249             'reason': set([('task_struct', 'personality')])},
4250            {'call': 'vfork',
4251             'reason': set([('task_struct', 'personality')])},
4252            {'call': 'io_setup', 'reason': set([('mm_struct', 'map_count')])},
4253            {'call': 'mprotect',
4254             'reason': set([('vm_area_struct', 'vm_end'),
4255                            ('vm_area_struct', 'vm_flags'),
4256                            ('vm_area_struct', 'vm_pgoff'),
4257                            ('vm_area_struct', 'vm_start')])},
4258            {'call': 'io_destroy',
4259             'reason': set([('mm_struct', 'map_count')])},
4260            {'call': 'mbind', 'reason': set([('mm_struct', 'map_count')])},
4261            {'call': 'prctl',
4262             'reason': set([('mm_struct', 'map_count'),
4263                            ('task_struct', 'personality'),
4264                            ('vm_area_struct', 'vm_end'),
4265                            ('vm_area_struct', 'vm_flags'),
4266                            ('vm_area_struct', 'vm_pgoff'),
4267                            ('vm_area_struct', 'vm_start')])},
4268            {'call': 'move_pages',
4269             'reason': set([('mm_struct', 'map_count'),
4270                            ('task_struct', 'personality')])},
4271            {'call': 'modify_ldt',
4272             'reason': set([('mm_struct', 'map_count')])},
4273            {'call': 'munlock',
4274             'reason': set([('vm_area_struct', 'vm_end'),
4275                            ('vm_area_struct', 'vm_flags'),
4276                            ('vm_area_struct', 'vm_pgoff'),
4277                            ('vm_area_struct', 'vm_start')])},
4278            {'call': 'setpriority',
4279             'reason': set([('task_struct', 'personality')])},
4280            {'call': 'mincore',
4281             'reason': set([('mm_struct', 'map_count'),
4282                            ('vm_area_struct', 'vm_end'),
4283                            ('vm_area_struct', 'vm_flags'),
```



```
4284                            ('vm_area_struct', 'vm_pgoff'),
4285                            ('vm_area_struct', 'vm_start')])},
4286            {'call': 'clone',
4287             'reason': set([('task_struct', 'personality')])},
4288            {'call': 'msync',
4289             'reason': set([('mm_struct', 'map_count'),
4290                            ('vm_area_struct', 'vm_end'),
4291                            ('vm_area_struct', 'vm_flags'),
4292                            ('vm_area_struct', 'vm_pgoff'),
4293                            ('vm_area_struct', 'vm_start')])},
4294            {'call': 'sched_getparam',
4295             'reason': set([('task_struct', 'personality')])},
4296            {'call': 'io_cancel',
4297             'reason': set([('mm_struct', 'map_count')])},
4298            {'call': 'mlockall',
4299             'reason': set([('vm_area_struct', 'vm_end'),
4300                            ('vm_area_struct', 'vm_flags'),
4301                            ('vm_area_struct', 'vm_pgoff'),
4302                            ('vm_area_struct', 'vm_start')])}],
4303  'msgctl': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
4304             {'call': 'rt_sigtimedwait',
4305              'reason': set([('mm_segment_t', 'seg')])},
4306             {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
4307             {'call': 'msgrcv',
4308              'reason': set([('ipc_namespace', 'msg_ctlmnb'),
4309                             ('mm_segment_t', 'seg')])},
4310             {'call': 'mq_unlink',
4311              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4312             {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
4313             {'call': 'msgget',
4314              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4315             {'call': 'sched_getaffinity',
4316              'reason': set([('mm_segment_t', 'seg')])},
4317             {'call': 'sched_setparam',
4318              'reason': set([('mm_segment_t', 'seg')])},
4319             {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
4320             {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
4321             {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
4322             {'call': 'mq_timedreceive',
4323              'reason': set([('mm_segment_t', 'seg')])},
4324             {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
4325             {'call': 'sched_setaffinity',
4326              'reason': set([('mm_segment_t', 'seg')])},
4327             {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
4328             {'call': 'semtimedop',
4329              'reason': set([('ipc_namespace', 'msg_ctlmnb'),
4330                             ('mm_segment_t', 'seg')])},
4331             {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
4332             {'call': 'sched_rr_get_interval',
4333              'reason': set([('mm_segment_t', 'seg')])},
4334             {'call': 'semctl',
4335              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4336             {'call': 'shmget',
4337              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4338             {'call': 'rt_sigprocmask',
4339              'reason': set([('mm_segment_t', 'seg')])},
4340             {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
4341             {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
4342             {'call': 'sched_setattr',
4343              'reason': set([('mm_segment_t', 'seg')])},
4344             {'call': 'migrate_pages',
4345              'reason': set([('mm_segment_t', 'seg')])},
4346             {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
4347             {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
4348             {'call': 'semget',
4349              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
```

```
4350             {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
4351             {'call': 'shmat',
4352              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4353             {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
4354             {'call': 'perf_event_open',
4355              'reason': set([('mm_segment_t', 'seg')])},
4356             {'call': 'rt_sigaction',
4357              'reason': set([('mm_segment_t', 'seg')])},
4358             {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
4359             {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
4360             {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
4361             {'call': 'setns',
4362              'reason': set([('ipc_namespace', 'msg_ctlmnb'),
4363                             ('mm_segment_t', 'seg')])},
4364             {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
4365             {'call': 'get_robust_list',
4366              'reason': set([('mm_segment_t', 'seg')])},
4367             {'call': 'mq_timedsend',
4368              'reason': set([('mm_segment_t', 'seg')])},
4369             {'call': 'sched_getscheduler',
4370              'reason': set([('mm_segment_t', 'seg')])},
4371             {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
4372             {'call': 'shmctl',
4373              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4374             {'call': 'sched_getattr',
4375              'reason': set([('mm_segment_t', 'seg')])},
4376             {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
4377             {'call': 'sched_setscheduler',
4378              'reason': set([('mm_segment_t', 'seg')])},
4379             {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
4380             {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
4381             {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
4382             {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
4383             {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
4384             {'call': 'msgsnd',
4385              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4386             {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
4387             {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
4388             {'call': 'mq_open',
4389              'reason': set([('ipc_namespace', 'msg_ctlmnb')])},
4390             {'call': 'sched_getparam',
4391              'reason': set([('mm_segment_t', 'seg')])}],
4392  'msgrcv': [{'call': 'mq_timedreceive', 'reason': set([('msg_msg', 'm_ts')])},
4393             {'call': 'mq_timedsend', 'reason': set([('msg_msg', 'm_ts')])},
4394             {'call': 'msgsnd', 'reason': set([('msg_msg', 'm_ts')])}],
4395  'msgsnd': [{'call': 'msgrcv',
4396              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4397             {'call': 'mq_unlink',
4398              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4399             {'call': 'msgget',
4400              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4401             {'call': 'semtimedop',
4402              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4403             {'call': 'semctl',
4404              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4405             {'call': 'shmget',
4406              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4407             {'call': 'msgctl',
4408              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4409             {'call': 'semget',
4410              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4411             {'call': 'shmat',
4412              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4413             {'call': 'setns',
4414              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4415             {'call': 'shmctl',
```



```
4416              'reason': set([('ipc_namespace', 'msg_ctlmax')])},
4417             {'call': 'mq_open',
4418              'reason': set([('ipc_namespace', 'msg_ctlmax')])}],
4419  'msync': [{'call': 'remap_file_pages',
4420             'reason': set([('vm_area_struct', 'vm_flags'),
4421                            ('vm_area_struct', 'vm_start')])},
4422            {'call': 'shmdt',
4423             'reason': set([('vm_area_struct', 'vm_flags'),
4424                            ('vm_area_struct', 'vm_start')])},
4425            {'call': 'brk',
4426             'reason': set([('vm_area_struct', 'vm_flags'),
4427                            ('vm_area_struct', 'vm_start')])},
4428            {'call': 'get_mempolicy',
4429             'reason': set([('vm_area_struct', 'vm_flags'),
4430                            ('vm_area_struct', 'vm_start')])},
4431            {'call': 'munlockall',
4432             'reason': set([('vm_area_struct', 'vm_flags'),
4433                            ('vm_area_struct', 'vm_start')])},
4434            {'call': 'pkey_mprotect',
4435             'reason': set([('vm_area_struct', 'vm_flags'),
4436                            ('vm_area_struct', 'vm_start')])},
4437            {'call': 'madvise',
4438             'reason': set([('vm_area_struct', 'vm_flags'),
4439                            ('vm_area_struct', 'vm_start')])},
4440            {'call': 'mprotect',
4441             'reason': set([('vm_area_struct', 'vm_flags'),
4442                            ('vm_area_struct', 'vm_start')])},
4443            {'call': 'mremap',
4444             'reason': set([('vm_area_struct', 'vm_flags'),
4445                            ('vm_area_struct', 'vm_start')])},
4446            {'call': 'prctl',
4447             'reason': set([('vm_area_struct', 'vm_flags'),
4448                            ('vm_area_struct', 'vm_start')])},
4449            {'call': 'munlock',
4450             'reason': set([('vm_area_struct', 'vm_flags'),
4451                            ('vm_area_struct', 'vm_start')])},
4452            {'call': 'mincore',
4453             'reason': set([('vm_area_struct', 'vm_flags'),
4454                            ('vm_area_struct', 'vm_start')])},
4455            {'call': 'mlockall',
4456             'reason': set([('vm_area_struct', 'vm_flags'),
4457                            ('vm_area_struct', 'vm_start')])}],
4458  'munlock': [{'call': 'remap_file_pages',
4459               'reason': set([('vm_area_struct', 'vm_end'),
4460                              ('vm_area_struct', 'vm_start')])},
4461              {'call': 'shmdt',
4462               'reason': set([('vm_area_struct', 'vm_end'),
4463                              ('vm_area_struct', 'vm_start')])},
4464              {'call': 'brk',
4465               'reason': set([('vm_area_struct', 'vm_end'),
4466                              ('vm_area_struct', 'vm_start')])},
4467              {'call': 'get_mempolicy',
4468               'reason': set([('vm_area_struct', 'vm_end'),
4469                              ('vm_area_struct', 'vm_start')])},
4470              {'call': 'munlockall',
4471               'reason': set([('vm_area_struct', 'vm_end'),
4472                              ('vm_area_struct', 'vm_start')])},
4473              {'call': 'pkey_mprotect',
4474               'reason': set([('vm_area_struct', 'vm_end'),
4475                              ('vm_area_struct', 'vm_start')])},
4476              {'call': 'madvise',
4477               'reason': set([('vm_area_struct', 'vm_end'),
4478                              ('vm_area_struct', 'vm_start')])},
4479              {'call': 'mprotect',
4480               'reason': set([('vm_area_struct', 'vm_end'),
4481                              ('vm_area_struct', 'vm_start')])},
```

```
4482             {'call': 'mremap',
4483              'reason': set([('vm_area_struct', 'vm_end'),
4484                             ('vm_area_struct', 'vm_start')])},
4485             {'call': 'prctl',
4486              'reason': set([('vm_area_struct', 'vm_end'),
4487                             ('vm_area_struct', 'vm_start')])},
4488             {'call': 'mincore',
4489              'reason': set([('vm_area_struct', 'vm_end'),
4490                             ('vm_area_struct', 'vm_start')])},
4491             {'call': 'msync',
4492              'reason': set([('vm_area_struct', 'vm_end'),
4493                             ('vm_area_struct', 'vm_start')])},
4494             {'call': 'mlockall',
4495              'reason': set([('vm_area_struct', 'vm_end'),
4496                             ('vm_area_struct', 'vm_start')])}],
4497  'munlockall': [{'call': 'remap_file_pages',
4498                  'reason': set([('vm_area_struct', 'vm_end'),
4499                                 ('vm_area_struct', 'vm_start')])},
4500                 {'call': 'shmdt',
4501                  'reason': set([('vm_area_struct', 'vm_end'),
4502                                 ('vm_area_struct', 'vm_start')])},
4503                 {'call': 'brk',
4504                  'reason': set([('vm_area_struct', 'vm_end'),
4505                                 ('vm_area_struct', 'vm_start')])},
4506                 {'call': 'get_mempolicy',
4507                  'reason': set([('vm_area_struct', 'vm_end'),
4508                                 ('vm_area_struct', 'vm_start')])},
4509                 {'call': 'pkey_mprotect',
4510                  'reason': set([('vm_area_struct', 'vm_end'),
4511                                 ('vm_area_struct', 'vm_start')])},
4512                 {'call': 'madvise',
4513                  'reason': set([('vm_area_struct', 'vm_end'),
4514                                 ('vm_area_struct', 'vm_start')])},
4515                 {'call': 'mprotect',
4516                  'reason': set([('vm_area_struct', 'vm_end'),
4517                                 ('vm_area_struct', 'vm_start')])},
4518                 {'call': 'mremap',
4519                  'reason': set([('vm_area_struct', 'vm_end'),
4520                                 ('vm_area_struct', 'vm_start')])},
4521                 {'call': 'prctl',
4522                  'reason': set([('vm_area_struct', 'vm_end'),
4523                                 ('vm_area_struct', 'vm_start')])},
4524                 {'call': 'munlock',
4525                  'reason': set([('vm_area_struct', 'vm_end'),
4526                                 ('vm_area_struct', 'vm_start')])},
4527                 {'call': 'mincore',
4528                  'reason': set([('vm_area_struct', 'vm_end'),
4529                                 ('vm_area_struct', 'vm_start')])},
4530                 {'call': 'msync',
4531                  'reason': set([('vm_area_struct', 'vm_end'),
4532                                 ('vm_area_struct', 'vm_start')])},
4533                 {'call': 'mlockall',
4534                  'reason': set([('vm_area_struct', 'vm_end'),
4535                                 ('vm_area_struct', 'vm_start')])}],
4536  'nanosleep': [{'call': 'rt_sigtimedwait',
4537                 'reason': set([('timespec', 'tv_nsec'),
4538                                ('timespec', 'tv_sec')])},
4539                {'call': 'fadvise64_64',
4540                 'reason': set([('timespec', 'tv_nsec'),
4541                                ('timespec', 'tv_sec')])},
4542                {'call': 'mq_unlink',
4543                 'reason': set([('timespec', 'tv_nsec'),
4544                                ('timespec', 'tv_sec')])},
4545                {'call': 'swapoff',
4546                 'reason': set([('timespec', 'tv_nsec'),
4547                                ('timespec', 'tv_sec')])},
```



```
4548                {'call': 'fchmod',
4549                 'reason': set([('timespec', 'tv_nsec'),
4550                                ('timespec', 'tv_sec')])},
4551                {'call': 'memfd_create',
4552                 'reason': set([('timespec', 'tv_nsec'),
4553                                ('timespec', 'tv_sec')])},
4554                {'call': 'readlinkat',
4555                 'reason': set([('timespec', 'tv_nsec'),
4556                                ('timespec', 'tv_sec')])},
4557                {'call': 'io_getevents',
4558                 'reason': set([('timespec', 'tv_nsec'),
4559                                ('timespec', 'tv_sec')])},
4560                {'call': 'fchown',
4561                 'reason': set([('timespec', 'tv_nsec'),
4562                                ('timespec', 'tv_sec')])},
4563                {'call': 'mq_timedreceive',
4564                 'reason': set([('timespec', 'tv_nsec'),
4565                                ('timespec', 'tv_sec')])},
4566                {'call': 'utime',
4567                 'reason': set([('timespec', 'tv_nsec'),
4568                                ('timespec', 'tv_sec')])},
4569                {'call': 'semtimedop',
4570                 'reason': set([('timespec', 'tv_nsec'),
4571                                ('timespec', 'tv_sec')])},
4572                {'call': 'settimeofday',
4573                 'reason': set([('timespec', 'tv_nsec'),
4574                                ('timespec', 'tv_sec')])},
4575                {'call': 'sched_rr_get_interval',
4576                 'reason': set([('timespec', 'tv_nsec'),
4577                                ('timespec', 'tv_sec')])},
4578                {'call': 'timerfd_gettime',
4579                 'reason': set([('timespec', 'tv_nsec'),
4580                                ('timespec', 'tv_sec')])},
4581                {'call': 'pselect6',
4582                 'reason': set([('timespec', 'tv_nsec'),
4583                                ('timespec', 'tv_sec')])},
4584                {'call': 'uselib',
4585                 'reason': set([('timespec', 'tv_nsec'),
4586                                ('timespec', 'tv_sec')])},
4587                {'call': 'fchmodat',
4588                 'reason': set([('timespec', 'tv_nsec'),
4589                                ('timespec', 'tv_sec')])},
4590                {'call': 'inotify_add_watch',
4591                 'reason': set([('timespec', 'tv_nsec'),
4592                                ('timespec', 'tv_sec')])},
4593                {'call': 'timer_settime',
4594                 'reason': set([('timespec', 'tv_nsec'),
4595                                ('timespec', 'tv_sec')])},
4596                {'call': 'ftruncate',
4597                 'reason': set([('timespec', 'tv_nsec'),
4598                                ('timespec', 'tv_sec')])},
4599                {'call': 'timer_gettime',
4600                 'reason': set([('timespec', 'tv_nsec'),
4601                                ('timespec', 'tv_sec')])},
4602                {'call': 'ioctl',
4603                 'reason': set([('timespec', 'tv_nsec'),
4604                                ('timespec', 'tv_sec')])},
4605                {'call': 'linkat',
4606                 'reason': set([('timespec', 'tv_nsec'),
4607                                ('timespec', 'tv_sec')])},
4608                {'call': 'stime',
4609                 'reason': set([('timespec', 'tv_nsec'),
4610                                ('timespec', 'tv_sec')])},
4611                {'call': 'futimesat',
4612                 'reason': set([('timespec', 'tv_nsec'),
4613                                ('timespec', 'tv_sec')])},
```

```
4614                {'call': 'poll',
4615                 'reason': set([('timespec', 'tv_nsec'),
4616                                ('timespec', 'tv_sec')])},
4617                {'call': 'select',
4618                 'reason': set([('timespec', 'tv_nsec'),
4619                                ('timespec', 'tv_sec')])},
4620                {'call': 'unlink',
4621                 'reason': set([('timespec', 'tv_nsec'),
4622                                ('timespec', 'tv_sec')])},
4623                {'call': 'mq_getsetattr',
4624                 'reason': set([('timespec', 'tv_nsec'),
4625                                ('timespec', 'tv_sec')])},
4626                {'call': 'faccessat',
4627                 'reason': set([('timespec', 'tv_nsec'),
4628                                ('timespec', 'tv_sec')])},
4629                {'call': 'mq_timedsend',
4630                 'reason': set([('timespec', 'tv_nsec'),
4631                                ('timespec', 'tv_sec')])},
4632                {'call': 'swapon',
4633                 'reason': set([('timespec', 'tv_nsec'),
4634                                ('timespec', 'tv_sec')])},
4635                {'call': 'epoll_wait',
4636                 'reason': set([('timespec', 'tv_nsec'),
4637                                ('timespec', 'tv_sec')])},
4638                {'call': 'fchownat',
4639                 'reason': set([('timespec', 'tv_nsec'),
4640                                ('timespec', 'tv_sec')])},
4641                {'call': 'timerfd_settime',
4642                 'reason': set([('timespec', 'tv_nsec'),
4643                                ('timespec', 'tv_sec')])},
4644                {'call': 'mq_notify',
4645                 'reason': set([('timespec', 'tv_nsec'),
4646                                ('timespec', 'tv_sec')])},
4647                {'call': 'sendfile',
4648                 'reason': set([('timespec', 'tv_nsec'),
4649                                ('timespec', 'tv_sec')])},
4650                {'call': 'clock_nanosleep',
4651                 'reason': set([('timespec', 'tv_nsec'),
4652                                ('timespec', 'tv_sec')])},
4653                {'call': 'unlinkat',
4654                 'reason': set([('timespec', 'tv_nsec'),
4655                                ('timespec', 'tv_sec')])},
4656                {'call': 'futex',
4657                 'reason': set([('timespec', 'tv_nsec'),
4658                                ('timespec', 'tv_sec')])},
4659                {'call': 'recvmmsg',
4660                 'reason': set([('timespec', 'tv_nsec'),
4661                                ('timespec', 'tv_sec')])},
4662                {'call': 'sendfile64',
4663                 'reason': set([('timespec', 'tv_nsec'),
4664                                ('timespec', 'tv_sec')])},
4665                {'call': 'ppoll',
4666                 'reason': set([('timespec', 'tv_nsec'),
4667                                ('timespec', 'tv_sec')])}],
4668  'newfstat': [{'call': 'newlstat',
4669                'reason': set([('compat_stat', 'st_ino'),
4670                               ('compat_stat', 'st_nlink'),
4671                               ('stat', 'st_ino'),
4672                               ('stat', 'st_nlink')])},
4673               {'call': 'newfstatat',
4674                'reason': set([('compat_stat', 'st_ino'),
4675                               ('compat_stat', 'st_nlink'),
4676                               ('stat', 'st_ino'),
4677                               ('stat', 'st_nlink')])},
4678               {'call': 'newstat',
4679                'reason': set([('compat_stat', 'st_ino'),
```



```
4680                               ('compat_stat', 'st_nlink'),
4681                               ('stat', 'st_ino'),
4682                               ('stat', 'st_nlink')])}],
4683  'newfstatat': [{'call': 'newlstat',
4684                  'reason': set([('compat_stat', 'st_ino'),
4685                                 ('compat_stat', 'st_nlink'),
4686                                 ('stat', 'st_ino'),
4687                                 ('stat', 'st_nlink')])},
4688                 {'call': 'newstat',
4689                  'reason': set([('compat_stat', 'st_ino'),
4690                                 ('compat_stat', 'st_nlink'),
4691                                 ('stat', 'st_ino'),
4692                                 ('stat', 'st_nlink')])},
4693                 {'call': 'newfstat',
4694                  'reason': set([('compat_stat', 'st_ino'),
4695                                 ('compat_stat', 'st_nlink'),
4696                                 ('stat', 'st_ino'),
4697                                 ('stat', 'st_nlink')])}],
4698  'newlstat': [{'call': 'newfstatat',
4699                'reason': set([('compat_stat', 'st_ino'),
4700                               ('compat_stat', 'st_nlink'),
4701                               ('stat', 'st_ino'),
4702                               ('stat', 'st_nlink')])},
4703               {'call': 'newstat',
4704                'reason': set([('compat_stat', 'st_ino'),
4705                               ('compat_stat', 'st_nlink'),
4706                               ('stat', 'st_ino'),
4707                               ('stat', 'st_nlink')])},
4708               {'call': 'newfstat',
4709                'reason': set([('compat_stat', 'st_ino'),
4710                               ('compat_stat', 'st_nlink'),
4711                               ('stat', 'st_ino'),
4712                               ('stat', 'st_nlink')])}],
4713  'newstat': [{'call': 'newlstat',
4714               'reason': set([('compat_stat', 'st_ino'),
4715                              ('compat_stat', 'st_nlink'),
4716                              ('stat', 'st_ino'),
4717                              ('stat', 'st_nlink')])},
4718              {'call': 'newfstatat',
4719               'reason': set([('compat_stat', 'st_ino'),
4720                              ('compat_stat', 'st_nlink'),
4721                              ('stat', 'st_ino'),
4722                              ('stat', 'st_nlink')])},
4723              {'call': 'newfstat',
4724               'reason': set([('compat_stat', 'st_ino'),
4725                              ('compat_stat', 'st_nlink'),
4726                              ('stat', 'st_ino'),
4727                              ('stat', 'st_nlink')])}],
4728  'newuname': [{'call': 'keyctl',
4729                'reason': set([('task_struct', 'personality')])},
4730               {'call': 'rt_sigtimedwait',
4731                'reason': set([('task_struct', 'personality')])},
4732               {'call': 'msgrcv',
4733                'reason': set([('task_struct', 'personality')])},
4734               {'call': 'kill',
4735                'reason': set([('task_struct', 'personality')])},
4736               {'call': 'sched_getaffinity',
4737                'reason': set([('task_struct', 'personality')])},
4738               {'call': 'sched_setparam',
4739                'reason': set([('task_struct', 'personality')])},
4740               {'call': 'ioprio_set',
4741                'reason': set([('task_struct', 'personality')])},
4742               {'call': 'personality',
4743                'reason': set([('task_struct', 'personality')])},
4744               {'call': 'getppid',
4745                'reason': set([('task_struct', 'personality')])},
```

```
4746                {'call': 'mq_timedreceive',
4747                 'reason': set([('task_struct', 'personality')])},
4748                {'call': 'capget',
4749                 'reason': set([('task_struct', 'personality')])},
4750                {'call': 'sched_setaffinity',
4751                 'reason': set([('task_struct', 'personality')])},
4752                {'call': 'signal',
4753                 'reason': set([('task_struct', 'personality')])},
4754                {'call': 'semtimedop',
4755                 'reason': set([('task_struct', 'personality')])},
4756                {'call': 'umount',
4757                 'reason': set([('task_struct', 'personality')])},
4758                {'call': 'sched_rr_get_interval',
4759                 'reason': set([('task_struct', 'personality')])},
4760                {'call': 'rt_sigprocmask',
4761                 'reason': set([('task_struct', 'personality')])},
4762                {'call': 'setsid',
4763                 'reason': set([('task_struct', 'personality')])},
4764                {'call': 'sigaltstack',
4765                 'reason': set([('task_struct', 'personality')])},
4766                {'call': 'sched_setattr',
4767                 'reason': set([('task_struct', 'personality')])},
4768                {'call': 'migrate_pages',
4769                 'reason': set([('task_struct', 'personality')])},
4770                {'call': 'getitimer',
4771                 'reason': set([('task_struct', 'personality')])},
4772                {'call': 'setpgid',
4773                 'reason': set([('task_struct', 'personality')])},
4774                {'call': 'getsid',
4775                 'reason': set([('task_struct', 'personality')])},
4776                {'call': 'prlimit64',
4777                 'reason': set([('task_struct', 'personality')])},
4778                {'call': 'perf_event_open',
4779                 'reason': set([('task_struct', 'personality')])},
4780                {'call': 'rt_sigaction',
4781                 'reason': set([('task_struct', 'personality')])},
4782                {'call': 'getpgid',
4783                 'reason': set([('task_struct', 'personality')])},
4784                {'call': 'getpriority',
4785                 'reason': set([('task_struct', 'personality')])},
4786                {'call': 'sigaction',
4787                 'reason': set([('task_struct', 'personality')])},
4788                {'call': 'setns',
4789                 'reason': set([('task_struct', 'personality')])},
4790                {'call': 'fork',
4791                 'reason': set([('task_struct', 'personality')])},
4792                {'call': 'get_robust_list',
4793                 'reason': set([('task_struct', 'personality')])},
4794                {'call': 'mq_timedsend',
4795                 'reason': set([('task_struct', 'personality')])},
4796                {'call': 'sched_getscheduler',
4797                 'reason': set([('task_struct', 'personality')])},
4798                {'call': 'ptrace',
4799                 'reason': set([('task_struct', 'personality')])},
4800                {'call': 'sched_getattr',
4801                 'reason': set([('task_struct', 'personality')])},
4802                {'call': 'getrusage',
4803                 'reason': set([('task_struct', 'personality')])},
4804                {'call': 'sched_setscheduler',
4805                 'reason': set([('task_struct', 'personality')])},
4806                {'call': 'setitimer',
4807                 'reason': set([('task_struct', 'personality')])},
4808                {'call': 'ioprio_get',
4809                 'reason': set([('task_struct', 'personality')])},
4810                {'call': 'vfork',
4811                 'reason': set([('task_struct', 'personality')])},
```



```
4812                {'call': 'prctl',
4813                 'reason': set([('task_struct', 'personality')])},
4814                {'call': 'move_pages',
4815                 'reason': set([('task_struct', 'personality')])},
4816                {'call': 'setpriority',
4817                 'reason': set([('task_struct', 'personality')])},
4818                {'call': 'clone',
4819                 'reason': set([('task_struct', 'personality')])},
4820                {'call': 'sched_getparam',
4821                 'reason': set([('task_struct', 'personality')])}],
4822  'old_getrlimit': [{'call': 'setrlimit',
4823                     'reason': set([('rlimit', 'rlim_cur'),
4824                                    ('rlimit', 'rlim_max')])},
4825                    {'call': 'prlimit64',
4826                     'reason': set([('rlimit', 'rlim_cur'),
4827                                    ('rlimit', 'rlim_max')])}],
4828  'old_readdir': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
4829                  {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
4830                  {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
4831                  {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
4832                  {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
4833                  {'call': 'readahead', 'reason': set([('fd', 'flags')])},
4834                  {'call': 'getdents', 'reason': set([('fd', 'flags')])},
4835                  {'call': 'writev', 'reason': set([('fd', 'flags')])},
4836                  {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
4837                  {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
4838                  {'call': 'pread64', 'reason': set([('fd', 'flags')])},
4839                  {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
4840                  {'call': 'read', 'reason': set([('fd', 'flags')])},
4841                  {'call': 'fchown', 'reason': set([('fd', 'flags')])},
4842                  {'call': 'mq_timedreceive',
4843                   'reason': set([('fd', 'flags')])},
4844                  {'call': 'utime', 'reason': set([('fd', 'flags')])},
4845                  {'call': 'fsync', 'reason': set([('fd', 'flags')])},
4846                  {'call': 'bpf', 'reason': set([('fd', 'flags')])},
4847                  {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
4848                  {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
4849                  {'call': 'sendto', 'reason': set([('fd', 'flags')])},
4850                  {'call': 'tee', 'reason': set([('fd', 'flags')])},
4851                  {'call': 'sync_file_range',
4852                   'reason': set([('fd', 'flags')])},
4853                  {'call': 'lseek', 'reason': set([('fd', 'flags')])},
4854                  {'call': 'connect', 'reason': set([('fd', 'flags')])},
4855                  {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
4856                  {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
4857                  {'call': 'flock', 'reason': set([('fd', 'flags')])},
4858                  {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
4859                  {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
4860                  {'call': 'accept4', 'reason': set([('fd', 'flags')])},
4861                  {'call': 'inotify_rm_watch',
4862                   'reason': set([('fd', 'flags')])},
4863                  {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
4864                  {'call': 'inotify_add_watch',
4865                   'reason': set([('fd', 'flags')])},
4866                  {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
4867                  {'call': 'splice', 'reason': set([('fd', 'flags')])},
4868                  {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
4869                  {'call': 'preadv', 'reason': set([('fd', 'flags')])},
4870                  {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
4871                  {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
4872                  {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
4873                  {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
4874                  {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
4875                  {'call': 'perf_event_open',
4876                   'reason': set([('fd', 'flags')])},
4877                  {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
```

```
4878                 {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
4879                 {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
4880                 {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
4881                 {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
4882                 {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
4883                 {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
4884                 {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
4885                 {'call': 'listen', 'reason': set([('fd', 'flags')])},
4886                 {'call': 'copy_file_range',
4887                  'reason': set([('fd', 'flags')])},
4888                 {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
4889                 {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
4890                 {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
4891                 {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
4892                 {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
4893                 {'call': 'llseek', 'reason': set([('fd', 'flags')])},
4894                 {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
4895                 {'call': 'readv', 'reason': set([('fd', 'flags')])},
4896                 {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
4897                 {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
4898                 {'call': 'write', 'reason': set([('fd', 'flags')])},
4899                 {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
4900                 {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
4901                 {'call': 'bind', 'reason': set([('fd', 'flags')])},
4902                 {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
4903                 {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
4904  'oldname': [{'call': 'keyctl',
4905               'reason': set([('mm_segment_t', 'seg'),
4906                              ('task_struct', 'personality')])},
4907              {'call': 'rt_sigtimedwait',
4908               'reason': set([('mm_segment_t', 'seg'),
4909                              ('task_struct', 'personality')])},
4910              {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
4911              {'call': 'msgrcv',
4912               'reason': set([('mm_segment_t', 'seg'),
4913                              ('task_struct', 'personality')])},
4914              {'call': 'kill',
4915               'reason': set([('mm_segment_t', 'seg'),
4916                              ('task_struct', 'personality')])},
4917              {'call': 'sched_getaffinity',
4918               'reason': set([('mm_segment_t', 'seg'),
4919                              ('task_struct', 'personality')])},
4920              {'call': 'sched_setparam',
4921               'reason': set([('mm_segment_t', 'seg'),
4922                              ('task_struct', 'personality')])},
4923              {'call': 'ioprio_set',
4924               'reason': set([('mm_segment_t', 'seg'),
4925                              ('task_struct', 'personality')])},
4926              {'call': 'personality',
4927               'reason': set([('task_struct', 'personality')])},
4928              {'call': 'getppid',
4929               'reason': set([('mm_segment_t', 'seg'),
4930                              ('task_struct', 'personality')])},
4931              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
4932              {'call': 'mq_timedreceive',
4933               'reason': set([('mm_segment_t', 'seg'),
4934                              ('task_struct', 'personality')])},
4935              {'call': 'capget',
4936               'reason': set([('mm_segment_t', 'seg'),
4937                              ('task_struct', 'personality')])},
4938              {'call': 'sched_setaffinity',
4939               'reason': set([('mm_segment_t', 'seg'),
4940                              ('task_struct', 'personality')])},
4941              {'call': 'signal',
4942               'reason': set([('mm_segment_t', 'seg'),
4943                              ('task_struct', 'personality')])},
```



```
4944              {'call': 'semtimedop',
4945               'reason': set([('mm_segment_t', 'seg'),
4946                              ('task_struct', 'personality')])},
4947              {'call': 'umount',
4948               'reason': set([('mm_segment_t', 'seg'),
4949                              ('task_struct', 'personality')])},
4950              {'call': 'sched_rr_get_interval',
4951               'reason': set([('mm_segment_t', 'seg'),
4952                              ('task_struct', 'personality')])},
4953              {'call': 'rt_sigprocmask',
4954               'reason': set([('mm_segment_t', 'seg'),
4955                              ('task_struct', 'personality')])},
4956              {'call': 'setsid',
4957               'reason': set([('mm_segment_t', 'seg'),
4958                              ('task_struct', 'personality')])},
4959              {'call': 'sigaltstack',
4960               'reason': set([('mm_segment_t', 'seg'),
4961                              ('task_struct', 'personality')])},
4962              {'call': 'sched_setattr',
4963               'reason': set([('mm_segment_t', 'seg'),
4964                              ('task_struct', 'personality')])},
4965              {'call': 'migrate_pages',
4966               'reason': set([('mm_segment_t', 'seg'),
4967                              ('task_struct', 'personality')])},
4968              {'call': 'getitimer',
4969               'reason': set([('mm_segment_t', 'seg'),
4970                              ('task_struct', 'personality')])},
4971              {'call': 'setpgid',
4972               'reason': set([('mm_segment_t', 'seg'),
4973                              ('task_struct', 'personality')])},
4974              {'call': 'getsid',
4975               'reason': set([('mm_segment_t', 'seg'),
4976                              ('task_struct', 'personality')])},
4977              {'call': 'prlimit64',
4978               'reason': set([('mm_segment_t', 'seg'),
4979                              ('task_struct', 'personality')])},
4980              {'call': 'perf_event_open',
4981               'reason': set([('mm_segment_t', 'seg'),
4982                              ('task_struct', 'personality')])},
4983              {'call': 'rt_sigaction',
4984               'reason': set([('mm_segment_t', 'seg'),
4985                              ('task_struct', 'personality')])},
4986              {'call': 'getpgid',
4987               'reason': set([('mm_segment_t', 'seg'),
4988                              ('task_struct', 'personality')])},
4989              {'call': 'getpriority',
4990               'reason': set([('mm_segment_t', 'seg'),
4991                              ('task_struct', 'personality')])},
4992              {'call': 'sigaction',
4993               'reason': set([('mm_segment_t', 'seg'),
4994                              ('task_struct', 'personality')])},
4995              {'call': 'setns',
4996               'reason': set([('mm_segment_t', 'seg'),
4997                              ('task_struct', 'personality')])},
4998              {'call': 'fork',
4999               'reason': set([('mm_segment_t', 'seg'),
5000                              ('task_struct', 'personality')])},
5001              {'call': 'get_robust_list',
5002               'reason': set([('mm_segment_t', 'seg'),
5003                              ('task_struct', 'personality')])},
5004              {'call': 'mq_timedsend',
5005               'reason': set([('mm_segment_t', 'seg'),
5006                              ('task_struct', 'personality')])},
5007              {'call': 'sched_getscheduler',
5008               'reason': set([('mm_segment_t', 'seg'),
5009                              ('task_struct', 'personality')])},
```

```
5010                {'call': 'ptrace',
5011                 'reason': set([('mm_segment_t', 'seg'),
5012                                ('task_struct', 'personality')])},
5013                {'call': 'sched_getattr',
5014                 'reason': set([('mm_segment_t', 'seg'),
5015                                ('task_struct', 'personality')])},
5016                {'call': 'getrusage',
5017                 'reason': set([('mm_segment_t', 'seg'),
5018                                ('task_struct', 'personality')])},
5019                {'call': 'sched_setscheduler',
5020                 'reason': set([('mm_segment_t', 'seg'),
5021                                ('task_struct', 'personality')])},
5022                {'call': 'setitimer',
5023                 'reason': set([('mm_segment_t', 'seg'),
5024                                ('task_struct', 'personality')])},
5025                {'call': 'ioprio_get',
5026                 'reason': set([('mm_segment_t', 'seg'),
5027                                ('task_struct', 'personality')])},
5028                {'call': 'vfork',
5029                 'reason': set([('mm_segment_t', 'seg'),
5030                                ('task_struct', 'personality')])},
5031                {'call': 'prctl',
5032                 'reason': set([('mm_segment_t', 'seg'),
5033                                ('task_struct', 'personality')])},
5034                {'call': 'move_pages',
5035                 'reason': set([('mm_segment_t', 'seg'),
5036                                ('task_struct', 'personality')])},
5037                {'call': 'setpriority',
5038                 'reason': set([('mm_segment_t', 'seg'),
5039                                ('task_struct', 'personality')])},
5040                {'call': 'clone',
5041                 'reason': set([('mm_segment_t', 'seg'),
5042                                ('task_struct', 'personality')])},
5043                {'call': 'sched_getparam',
5044                 'reason': set([('mm_segment_t', 'seg'),
5045                                ('task_struct', 'personality')])}],
5046  'perf_event_open': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
5047                      {'call': 'keyctl',
5048                       'reason': set([('mm_segment_t', 'seg'),
5049                                      ('task_struct', 'flags')])},
5050                      {'call': 'rt_sigtimedwait',
5051                       'reason': set([('mm_segment_t', 'seg'),
5052                                      ('task_struct', 'flags')])},
5053                      {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
5054                      {'call': 'iopl',
5055                       'reason': set([('mm_segment_t', 'seg')])},
5056                      {'call': 'msgrcv',
5057                       'reason': set([('mm_segment_t', 'seg'),
5058                                      ('task_struct', 'flags')])},
5059                      {'call': 'fadvise64_64',
5060                       'reason': set([('fd', 'flags')])},
5061                      {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
5062                      {'call': 'kill',
5063                       'reason': set([('mm_segment_t', 'seg'),
5064                                      ('task_struct', 'flags')])},
5065                      {'call': 'fremovexattr',
5066                       'reason': set([('fd', 'flags')])},
5067                      {'call': 'readahead', 'reason': set([('fd', 'flags')])},
5068                      {'call': 'getdents', 'reason': set([('fd', 'flags')])},
5069                      {'call': 'sched_getaffinity',
5070                       'reason': set([('mm_segment_t', 'seg'),
5071                                      ('task_struct', 'flags')])},
5072                      {'call': 'writev', 'reason': set([('fd', 'flags')])},
5073                      {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
5074                      {'call': 'sched_setparam',
5075                       'reason': set([('mm_segment_t', 'seg'),
```



```
5076                                      ('task_struct', 'flags')])},
5077                      {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
5078                      {'call': 'pread64', 'reason': set([('fd', 'flags')])},
5079                      {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
5080                      {'call': 'ioprio_set',
5081                       'reason': set([('mm_segment_t', 'seg'),
5082                                      ('task_struct', 'flags')])},
5083                      {'call': 'read', 'reason': set([('fd', 'flags')])},
5084                      {'call': 'getppid',
5085                       'reason': set([('mm_segment_t', 'seg'),
5086                                      ('task_struct', 'flags')])},
5087                      {'call': 'fchown', 'reason': set([('fd', 'flags')])},
5088                      {'call': 'ioperm',
5089                       'reason': set([('mm_segment_t', 'seg')])},
5090                      {'call': 'mq_timedreceive',
5091                       'reason': set([('fd', 'flags'),
5092                                      ('mm_segment_t', 'seg'),
5093                                      ('task_struct', 'flags')])},
5094                      {'call': 'capget',
5095                       'reason': set([('mm_segment_t', 'seg'),
5096                                      ('task_struct', 'flags')])},
5097                      {'call': 'utime', 'reason': set([('fd', 'flags')])},
5098                      {'call': 'sched_setaffinity',
5099                       'reason': set([('mm_segment_t', 'seg'),
5100                                      ('task_struct', 'flags')])},
5101                      {'call': 'fsync', 'reason': set([('fd', 'flags')])},
5102                      {'call': 'bpf', 'reason': set([('fd', 'flags')])},
5103                      {'call': 'signal',
5104                       'reason': set([('mm_segment_t', 'seg'),
5105                                      ('task_struct', 'flags')])},
5106                      {'call': 'setreuid',
5107                       'reason': set([('task_struct', 'flags')])},
5108                      {'call': 'semtimedop',
5109                       'reason': set([('mm_segment_t', 'seg'),
5110                                      ('task_struct', 'flags')])},
5111                      {'call': 'umount',
5112                       'reason': set([('mm_segment_t', 'seg'),
5113                                      ('task_struct', 'flags')])},
5114                      {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
5115                      {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
5116                      {'call': 'sendto', 'reason': set([('fd', 'flags')])},
5117                      {'call': 'sched_rr_get_interval',
5118                       'reason': set([('mm_segment_t', 'seg'),
5119                                      ('task_struct', 'flags')])},
5120                      {'call': 'tee', 'reason': set([('fd', 'flags')])},
5121                      {'call': 'sync_file_range',
5122                       'reason': set([('fd', 'flags')])},
5123                      {'call': 'lseek', 'reason': set([('fd', 'flags')])},
5124                      {'call': 'connect', 'reason': set([('fd', 'flags')])},
5125                      {'call': 'getsockname',
5126                       'reason': set([('fd', 'flags')])},
5127                      {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
5128                      {'call': 'flock', 'reason': set([('fd', 'flags')])},
5129                      {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
5130                      {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
5131                      {'call': 'rt_sigprocmask',
5132                       'reason': set([('mm_segment_t', 'seg'),
5133                                      ('task_struct', 'flags')])},
5134                      {'call': 'accept4', 'reason': set([('fd', 'flags')])},
5135                      {'call': 'setsid',
5136                       'reason': set([('mm_segment_t', 'seg'),
5137                                      ('task_struct', 'flags')])},
5138                      {'call': 'sigaltstack',
5139                       'reason': set([('mm_segment_t', 'seg'),
5140                                      ('task_struct', 'flags')])},
5141                      {'call': 'sched_setattr',
```

```
5142                      'reason': set([('mm_segment_t', 'seg'),
5143                                     ('task_struct', 'flags')])},
5144                    {'call': 'old_readdir',
5145                     'reason': set([('fd', 'flags')])},
5146                    {'call': 'inotify_rm_watch',
5147                     'reason': set([('fd', 'flags')])},
5148                    {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
5149                    {'call': 'migrate_pages',
5150                     'reason': set([('mm_segment_t', 'seg'),
5151                                    ('task_struct', 'flags')])},
5152                    {'call': 'getitimer',
5153                     'reason': set([('mm_segment_t', 'seg'),
5154                                    ('task_struct', 'flags')])},
5155                    {'call': 'setpgid',
5156                     'reason': set([('mm_segment_t', 'seg'),
5157                                    ('task_struct', 'flags')])},
5158                    {'call': 'inotify_add_watch',
5159                     'reason': set([('fd', 'flags')])},
5160                    {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
5161                    {'call': 'splice', 'reason': set([('fd', 'flags')])},
5162                    {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
5163                    {'call': 'preadv', 'reason': set([('fd', 'flags')])},
5164                    {'call': 'getpeername',
5165                     'reason': set([('fd', 'flags')])},
5166                    {'call': 'getsid',
5167                     'reason': set([('mm_segment_t', 'seg'),
5168                                    ('task_struct', 'flags')])},
5169                    {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
5170                    {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
5171                    {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
5172                    {'call': 'prlimit64',
5173                     'reason': set([('mm_segment_t', 'seg'),
5174                                    ('task_struct', 'flags')])},
5175                    {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
5176                    {'call': 'pwritev64v2',
5177                     'reason': set([('fd', 'flags')])},
5178                    {'call': 'rt_sigaction',
5179                     'reason': set([('mm_segment_t', 'seg'),
5180                                    ('task_struct', 'flags')])},
5181                    {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
5182                    {'call': 'getpgid',
5183                     'reason': set([('mm_segment_t', 'seg'),
5184                                    ('task_struct', 'flags')])},
5185                    {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
5186                    {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
5187                    {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
5188                    {'call': 'getpriority',
5189                     'reason': set([('mm_segment_t', 'seg'),
5190                                    ('task_struct', 'flags')])},
5191                    {'call': 'sigaction',
5192                     'reason': set([('mm_segment_t', 'seg'),
5193                                    ('task_struct', 'flags')])},
5194                    {'call': 'mq_getsetattr',
5195                     'reason': set([('fd', 'flags')])},
5196                    {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
5197                    {'call': 'setns',
5198                     'reason': set([('mm_segment_t', 'seg'),
5199                                    ('task_struct', 'flags')])},
5200                    {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
5201                    {'call': 'listen', 'reason': set([('fd', 'flags')])},
5202                    {'call': 'fork',
5203                     'reason': set([('mm_segment_t', 'seg'),
5204                                    ('task_struct', 'flags')])},
5205                    {'call': 'get_robust_list',
5206                     'reason': set([('mm_segment_t', 'seg'),
5207                                    ('task_struct', 'flags')])},
```



```
5208                    {'call': 'copy_file_range',
5209                     'reason': set([('fd', 'flags')])},
5210                    {'call': 'mq_timedsend',
5211                     'reason': set([('fd', 'flags'),
5212                                    ('mm_segment_t', 'seg'),
5213                                    ('task_struct', 'flags')])},
5214                    {'call': 'sched_getscheduler',
5215                     'reason': set([('mm_segment_t', 'seg'),
5216                                    ('task_struct', 'flags')])},
5217                    {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
5218                    {'call': 'ptrace',
5219                     'reason': set([('mm_segment_t', 'seg'),
5220                                    ('task_struct', 'flags')])},
5221                    {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
5222                    {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
5223                    {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
5224                    {'call': 'sched_getattr',
5225                     'reason': set([('mm_segment_t', 'seg'),
5226                                    ('task_struct', 'flags')])},
5227                    {'call': 'getrusage',
5228                     'reason': set([('mm_segment_t', 'seg'),
5229                                    ('task_struct', 'flags')])},
5230                    {'call': 'sched_setscheduler',
5231                     'reason': set([('mm_segment_t', 'seg'),
5232                                    ('task_struct', 'flags')])},
5233                    {'call': 'setresuid',
5234                     'reason': set([('task_struct', 'flags')])},
5235                    {'call': 'setitimer',
5236                     'reason': set([('mm_segment_t', 'seg'),
5237                                    ('task_struct', 'flags')])},
5238                    {'call': 'ioprio_get',
5239                     'reason': set([('mm_segment_t', 'seg'),
5240                                    ('task_struct', 'flags')])},
5241                    {'call': 'vfork',
5242                     'reason': set([('mm_segment_t', 'seg'),
5243                                    ('task_struct', 'flags')])},
5244                    {'call': 'setuid',
5245                     'reason': set([('task_struct', 'flags')])},
5246                    {'call': 'llseek', 'reason': set([('fd', 'flags')])},
5247                    {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
5248                    {'call': 'readv', 'reason': set([('fd', 'flags')])},
5249                    {'call': 'prctl',
5250                     'reason': set([('mm_segment_t', 'seg'),
5251                                    ('task_struct', 'flags')])},
5252                    {'call': 'move_pages',
5253                     'reason': set([('mm_segment_t', 'seg'),
5254                                    ('task_struct', 'flags')])},
5255                    {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
5256                    {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
5257                    {'call': 'write', 'reason': set([('fd', 'flags')])},
5258                    {'call': 'setpriority',
5259                     'reason': set([('mm_segment_t', 'seg'),
5260                                    ('task_struct', 'flags')])},
5261                    {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
5262                    {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
5263                    {'call': 'clone',
5264                     'reason': set([('mm_segment_t', 'seg'),
5265                                    ('task_struct', 'flags')])},
5266                    {'call': 'sched_getparam',
5267                     'reason': set([('mm_segment_t', 'seg'),
5268                                    ('task_struct', 'flags')])},
5269                    {'call': 'bind', 'reason': set([('fd', 'flags')])},
5270                    {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
5271                    {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
5272  'pivot_root': [{'call': 'mq_unlink',
5273                  'reason': set([('vfsmount', 'mnt_flags')])},
```

```
5274                {'call': 'umount',
5275                 'reason': set([('vfsmount', 'mnt_flags')])},
5276                {'call': 'getcwd',
5277                 'reason': set([('vfsmount', 'mnt_flags')])},
5278                {'call': 'acct', 'reason': set([('vfsmount', 'mnt_flags')])},
5279                {'call': 'mq_open',
5280                 'reason': set([('vfsmount', 'mnt_flags')])}],
5281  'pkey_alloc': [{'call': 'swapoff',
5282                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5283                 {'call': 'remap_file_pages',
5284                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5285                 {'call': 'io_getevents',
5286                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5287                 {'call': 'pkey_free',
5288                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5289                 {'call': 'migrate_pages',
5290                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5291                 {'call': 'shmdt',
5292                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5293                 {'call': 'brk',
5294                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5295                 {'call': 'get_mempolicy',
5296                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5297                 {'call': 'getrusage',
5298                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5299                 {'call': 'io_setup',
5300                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5301                 {'call': 'mremap',
5302                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5303                 {'call': 'io_destroy',
5304                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5305                 {'call': 'mbind',
5306                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5307                 {'call': 'prctl',
5308                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5309                 {'call': 'move_pages',
5310                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5311                 {'call': 'modify_ldt',
5312                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5313                 {'call': 'mincore',
5314                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5315                 {'call': 'msync',
5316                  'reason': set([('mm_context_t', 'pkey_allocation_map')])},
5317                 {'call': 'io_cancel',
5318                  'reason': set([('mm_context_t', 'pkey_allocation_map')])}],
5319  'pkey_mprotect': [{'call': 'keyctl',
5320                     'reason': set([('task_struct', 'personality')])},
5321                    {'call': 'rt_sigtimedwait',
5322                     'reason': set([('task_struct', 'personality')])},
5323                    {'call': 'msgrcv',
5324                     'reason': set([('task_struct', 'personality')])},
5325                    {'call': 'kill',
5326                     'reason': set([('task_struct', 'personality')])},
5327                    {'call': 'sched_getaffinity',
5328                     'reason': set([('task_struct', 'personality')])},
5329                    {'call': 'sched_setparam',
5330                     'reason': set([('task_struct', 'personality')])},
5331                    {'call': 'ioprio_set',
5332                     'reason': set([('task_struct', 'personality')])},
5333                    {'call': 'personality',
5334                     'reason': set([('task_struct', 'personality')])},
5335                    {'call': 'remap_file_pages',
5336                     'reason': set([('vm_area_struct', 'vm_end'),
5337                                    ('vm_area_struct', 'vm_flags'),
5338                                    ('vm_area_struct', 'vm_start')])},
5339                    {'call': 'getppid',
```



```
5340                     'reason': set([('task_struct', 'personality')])},
5341                    {'call': 'mq_timedreceive',
5342                     'reason': set([('task_struct', 'personality')])},
5343                    {'call': 'capget',
5344                     'reason': set([('task_struct', 'personality')])},
5345                    {'call': 'sched_setaffinity',
5346                     'reason': set([('task_struct', 'personality')])},
5347                    {'call': 'signal',
5348                     'reason': set([('task_struct', 'personality')])},
5349                    {'call': 'semtimedop',
5350                     'reason': set([('task_struct', 'personality')])},
5351                    {'call': 'umount',
5352                     'reason': set([('task_struct', 'personality')])},
5353                    {'call': 'sched_rr_get_interval',
5354                     'reason': set([('task_struct', 'personality')])},
5355                    {'call': 'rt_sigprocmask',
5356                     'reason': set([('task_struct', 'personality')])},
5357                    {'call': 'setsid',
5358                     'reason': set([('task_struct', 'personality')])},
5359                    {'call': 'sigaltstack',
5360                     'reason': set([('task_struct', 'personality')])},
5361                    {'call': 'sched_setattr',
5362                     'reason': set([('task_struct', 'personality')])},
5363                    {'call': 'migrate_pages',
5364                     'reason': set([('task_struct', 'personality')])},
5365                    {'call': 'getitimer',
5366                     'reason': set([('task_struct', 'personality')])},
5367                    {'call': 'setpgid',
5368                     'reason': set([('task_struct', 'personality')])},
5369                    {'call': 'getsid',
5370                     'reason': set([('task_struct', 'personality')])},
5371                    {'call': 'prlimit64',
5372                     'reason': set([('task_struct', 'personality')])},
5373                    {'call': 'perf_event_open',
5374                     'reason': set([('task_struct', 'personality')])},
5375                    {'call': 'shmdt',
5376                     'reason': set([('vm_area_struct', 'vm_end'),
5377                                    ('vm_area_struct', 'vm_flags'),
5378                                    ('vm_area_struct', 'vm_start')])},
5379                    {'call': 'rt_sigaction',
5380                     'reason': set([('task_struct', 'personality')])},
5381                    {'call': 'getpgid',
5382                     'reason': set([('task_struct', 'personality')])},
5383                    {'call': 'brk',
5384                     'reason': set([('vm_area_struct', 'vm_end'),
5385                                    ('vm_area_struct', 'vm_flags'),
5386                                    ('vm_area_struct', 'vm_start')])},
5387                    {'call': 'getpriority',
5388                     'reason': set([('task_struct', 'personality')])},
5389                    {'call': 'sigaction',
5390                     'reason': set([('task_struct', 'personality')])},
5391                    {'call': 'setns',
5392                     'reason': set([('task_struct', 'personality')])},
5393                    {'call': 'fork',
5394                     'reason': set([('task_struct', 'personality')])},
5395                    {'call': 'get_mempolicy',
5396                     'reason': set([('vm_area_struct', 'vm_end'),
5397                                    ('vm_area_struct', 'vm_flags'),
5398                                    ('vm_area_struct', 'vm_start')])},
5399                    {'call': 'get_robust_list',
5400                     'reason': set([('task_struct', 'personality')])},
5401                    {'call': 'mq_timedsend',
5402                     'reason': set([('task_struct', 'personality')])},
5403                    {'call': 'sched_getscheduler',
5404                     'reason': set([('task_struct', 'personality')])},
5405                    {'call': 'ptrace',
```

```
5406                     'reason': set([('task_struct', 'personality')])},
5407                    {'call': 'munlockall',
5408                     'reason': set([('vm_area_struct', 'vm_end'),
5409                                    ('vm_area_struct', 'vm_flags'),
5410                                    ('vm_area_struct', 'vm_start')])},
5411                    {'call': 'madvise',
5412                     'reason': set([('vm_area_struct', 'vm_end'),
5413                                    ('vm_area_struct', 'vm_flags'),
5414                                    ('vm_area_struct', 'vm_start')])},
5415                    {'call': 'sched_getattr',
5416                     'reason': set([('task_struct', 'personality')])},
5417                    {'call': 'getrusage',
5418                     'reason': set([('task_struct', 'personality')])},
5419                    {'call': 'sched_setscheduler',
5420                     'reason': set([('task_struct', 'personality')])},
5421                    {'call': 'setitimer',
5422                     'reason': set([('task_struct', 'personality')])},
5423                    {'call': 'ioprio_get',
5424                     'reason': set([('task_struct', 'personality')])},
5425                    {'call': 'vfork',
5426                     'reason': set([('task_struct', 'personality')])},
5427                    {'call': 'mprotect',
5428                     'reason': set([('vm_area_struct', 'vm_end'),
5429                                    ('vm_area_struct', 'vm_flags'),
5430                                    ('vm_area_struct', 'vm_start')])},
5431                    {'call': 'mremap',
5432                     'reason': set([('vm_area_struct', 'vm_end'),
5433                                    ('vm_area_struct', 'vm_flags'),
5434                                    ('vm_area_struct', 'vm_start')])},
5435                    {'call': 'prctl',
5436                     'reason': set([('task_struct', 'personality'),
5437                                    ('vm_area_struct', 'vm_end'),
5438                                    ('vm_area_struct', 'vm_flags'),
5439                                    ('vm_area_struct', 'vm_start')])},
5440                    {'call': 'move_pages',
5441                     'reason': set([('task_struct', 'personality')])},
5442                    {'call': 'munlock',
5443                     'reason': set([('vm_area_struct', 'vm_end'),
5444                                    ('vm_area_struct', 'vm_flags'),
5445                                    ('vm_area_struct', 'vm_start')])},
5446                    {'call': 'setpriority',
5447                     'reason': set([('task_struct', 'personality')])},
5448                    {'call': 'mincore',
5449                     'reason': set([('vm_area_struct', 'vm_end'),
5450                                    ('vm_area_struct', 'vm_flags'),
5451                                    ('vm_area_struct', 'vm_start')])},
5452                    {'call': 'clone',
5453                     'reason': set([('task_struct', 'personality')])},
5454                    {'call': 'msync',
5455                     'reason': set([('vm_area_struct', 'vm_end'),
5456                                    ('vm_area_struct', 'vm_flags'),
5457                                    ('vm_area_struct', 'vm_start')])},
5458                    {'call': 'sched_getparam',
5459                     'reason': set([('task_struct', 'personality')])},
5460                    {'call': 'mlockall',
5461                     'reason': set([('vm_area_struct', 'vm_end'),
5462                                    ('vm_area_struct', 'vm_flags'),
5463                                    ('vm_area_struct', 'vm_start')])}],
5464  'poll': [{'call': 'ppoll', 'reason': set([('poll_list', 'len')])}],
5465  'ppoll': [{'call': 'keyctl',
5466             'reason': set([('task_struct', 'personality')])},
5467            {'call': 'rt_sigtimedwait',
5468             'reason': set([('task_struct', 'personality'),
5469                            ('timespec', 'tv_nsec'),
5470                            ('timespec', 'tv_sec')])},
5471            {'call': 'msgrcv',
```



```
5472             'reason': set([('task_struct', 'personality')])},
5473            {'call': 'fadvise64_64',
5474             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5475            {'call': 'mq_unlink',
5476             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5477            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
5478            {'call': 'swapoff',
5479             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5480            {'call': 'sched_getaffinity',
5481             'reason': set([('task_struct', 'personality')])},
5482            {'call': 'sched_setparam',
5483             'reason': set([('task_struct', 'personality')])},
5484            {'call': 'fchmod',
5485             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5486            {'call': 'memfd_create',
5487             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5488            {'call': 'ioprio_set',
5489             'reason': set([('task_struct', 'personality')])},
5490            {'call': 'personality',
5491             'reason': set([('task_struct', 'personality')])},
5492            {'call': 'readlinkat',
5493             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5494            {'call': 'io_getevents',
5495             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5496            {'call': 'getppid',
5497             'reason': set([('task_struct', 'personality')])},
5498            {'call': 'fchown',
5499             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5500            {'call': 'mq_timedreceive',
5501             'reason': set([('task_struct', 'personality'),
5502                            ('timespec', 'tv_nsec'),
5503                            ('timespec', 'tv_sec')])},
5504            {'call': 'capget',
5505             'reason': set([('task_struct', 'personality')])},
5506            {'call': 'utime',
5507             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5508            {'call': 'sched_setaffinity',
5509             'reason': set([('task_struct', 'personality')])},
5510            {'call': 'signal',
5511             'reason': set([('task_struct', 'personality')])},
5512            {'call': 'semtimedop',
5513             'reason': set([('task_struct', 'personality'),
5514                            ('timespec', 'tv_nsec'),
5515                            ('timespec', 'tv_sec')])},
5516            {'call': 'umount',
5517             'reason': set([('task_struct', 'personality')])},
5518            {'call': 'settimeofday',
5519             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5520            {'call': 'sched_rr_get_interval',
5521             'reason': set([('task_struct', 'personality'),
5522                            ('timespec', 'tv_nsec'),
5523                            ('timespec', 'tv_sec')])},
5524            {'call': 'timerfd_gettime',
5525             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5526            {'call': 'pselect6',
5527             'reason': set([('compat_timespec', 'tv_nsec'),
5528                            ('compat_timespec', 'tv_sec'),
5529                            ('timespec', 'tv_nsec'),
5530                            ('timespec', 'tv_sec')])},
5531            {'call': 'uselib',
5532             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5533            {'call': 'rt_sigprocmask',
5534             'reason': set([('task_struct', 'personality')])},
5535            {'call': 'setsid',
5536             'reason': set([('task_struct', 'personality')])},
5537            {'call': 'sigaltstack',
```

```
5538             'reason': set([('task_struct', 'personality')])},
5539            {'call': 'sched_setattr',
5540             'reason': set([('task_struct', 'personality')])},
5541            {'call': 'migrate_pages',
5542             'reason': set([('task_struct', 'personality')])},
5543            {'call': 'getitimer',
5544             'reason': set([('task_struct', 'personality')])},
5545            {'call': 'fchmodat',
5546             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5547            {'call': 'setpgid',
5548             'reason': set([('task_struct', 'personality')])},
5549            {'call': 'inotify_add_watch',
5550             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5551            {'call': 'timer_settime',
5552             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5553            {'call': 'ftruncate',
5554             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5555            {'call': 'timer_gettime',
5556             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5557            {'call': 'getsid',
5558             'reason': set([('task_struct', 'personality')])},
5559            {'call': 'ioctl',
5560             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5561            {'call': 'prlimit64',
5562             'reason': set([('task_struct', 'personality')])},
5563            {'call': 'perf_event_open',
5564             'reason': set([('task_struct', 'personality')])},
5565            {'call': 'linkat',
5566             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5567            {'call': 'stime',
5568             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5569            {'call': 'rt_sigaction',
5570             'reason': set([('task_struct', 'personality')])},
5571            {'call': 'futimesat',
5572             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5573            {'call': 'getpgid',
5574             'reason': set([('task_struct', 'personality')])},
5575            {'call': 'poll',
5576             'reason': set([('poll_list', 'len'),
5577                            ('timespec', 'tv_nsec'),
5578                            ('timespec', 'tv_sec')])},
5579            {'call': 'select',
5580             'reason': set([('compat_timespec', 'tv_nsec'),
5581                            ('compat_timespec', 'tv_sec'),
5582                            ('timespec', 'tv_nsec'),
5583                            ('timespec', 'tv_sec')])},
5584            {'call': 'unlink',
5585             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5586            {'call': 'getpriority',
5587             'reason': set([('task_struct', 'personality')])},
5588            {'call': 'sigaction',
5589             'reason': set([('task_struct', 'personality')])},
5590            {'call': 'nanosleep',
5591             'reason': set([('compat_timespec', 'tv_nsec'),
5592                            ('compat_timespec', 'tv_sec'),
5593                            ('timespec', 'tv_nsec'),
5594                            ('timespec', 'tv_sec')])},
5595            {'call': 'mq_getsetattr',
5596             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5597            {'call': 'faccessat',
5598             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5599            {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
5600            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
5601            {'call': 'get_robust_list',
5602             'reason': set([('task_struct', 'personality')])},
5603            {'call': 'mq_timedsend',
```



```
5604             'reason': set([('task_struct', 'personality'),
5605                            ('timespec', 'tv_nsec'),
5606                            ('timespec', 'tv_sec')])},
5607            {'call': 'sched_getscheduler',
5608             'reason': set([('task_struct', 'personality')])},
5609            {'call': 'ptrace',
5610             'reason': set([('task_struct', 'personality')])},
5611            {'call': 'swapon',
5612             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5613            {'call': 'epoll_wait',
5614             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5615            {'call': 'sched_getattr',
5616             'reason': set([('task_struct', 'personality')])},
5617            {'call': 'fchownat',
5618             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5619            {'call': 'getrusage',
5620             'reason': set([('task_struct', 'personality')])},
5621            {'call': 'timerfd_settime',
5622             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5623            {'call': 'sched_setscheduler',
5624             'reason': set([('task_struct', 'personality')])},
5625            {'call': 'setitimer',
5626             'reason': set([('task_struct', 'personality')])},
5627            {'call': 'ioprio_get',
5628             'reason': set([('task_struct', 'personality')])},
5629            {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
5630            {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
5631            {'call': 'move_pages',
5632             'reason': set([('task_struct', 'personality')])},
5633            {'call': 'setpriority',
5634             'reason': set([('task_struct', 'personality')])},
5635            {'call': 'mq_notify',
5636             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5637            {'call': 'sendfile',
5638             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5639            {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
5640            {'call': 'clock_nanosleep',
5641             'reason': set([('compat_timespec', 'tv_nsec'),
5642                            ('compat_timespec', 'tv_sec'),
5643                            ('timespec', 'tv_nsec'),
5644                            ('timespec', 'tv_sec')])},
5645            {'call': 'unlinkat',
5646             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5647            {'call': 'sched_getparam',
5648             'reason': set([('task_struct', 'personality')])},
5649            {'call': 'futex',
5650             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5651            {'call': 'recvmmsg',
5652             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])},
5653            {'call': 'sendfile64',
5654             'reason': set([('timespec', 'tv_nsec'), ('timespec', 'tv_sec')])}],
5655  'prctl': [{'call': 'keyctl',
5656             'reason': set([('task_struct', 'flags'),
5657                            ('task_struct', 'personality'),
5658                            ('task_struct', 'timer_slack_ns')])},
5659            {'call': 'rt_sigtimedwait',
5660             'reason': set([('task_struct', 'flags'),
5661                            ('task_struct', 'personality'),
5662                            ('task_struct', 'timer_slack_ns')])},
5663            {'call': 'msgrcv',
5664             'reason': set([('task_struct', 'flags'),
5665                            ('task_struct', 'personality'),
5666                            ('task_struct', 'timer_slack_ns')])},
5667            {'call': 'kill',
5668             'reason': set([('task_struct', 'flags'),
5669                            ('task_struct', 'personality'),
```

```
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'swapoff', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'sched_getaffinity',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_setparam',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'ioprio_set',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'personality',
           'reason': set([('task_struct', 'personality')])},
          {'call': 'remap_file_pages',
           'reason': set([('mm_struct', 'flags')])},
          {'call': 'io_getevents', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'getppid',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'mq_timedreceive',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'capget',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_setaffinity',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'signal',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
          {'call': 'semtimedop',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'umount',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_rr_get_interval',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'rt_sigprocmask',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'setsid',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'sigaltstack',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_setattr',
           'reason': set([('task_struct', 'flags'),
```



```
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'migrate_pages',
           'reason': set([('mm_struct', 'flags'),
                          ('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'getitimer',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'setpgid',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'getsid',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'prlimit64',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'perf_event_open',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'shmdt', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'rt_sigaction',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'getpgid',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'brk', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'getpriority',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'sigaction',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'setns',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'fork',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'get_mempolicy', 'reason': set([('mm_struct', 'flags')])},
          {'call': 'get_robust_list',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'mq_timedsend',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
          {'call': 'sched_getscheduler',
           'reason': set([('task_struct', 'flags'),
                          ('task_struct', 'personality'),
                          ('task_struct', 'timer_slack_ns')])},
```

```
5802            {'call': 'ptrace',
5803             'reason': set([('task_struct', 'flags'),
5804                            ('task_struct', 'personality'),
5805                            ('task_struct', 'timer_slack_ns')])},
5806            {'call': 'sched_getattr',
5807             'reason': set([('task_struct', 'flags'),
5808                            ('task_struct', 'personality'),
5809                            ('task_struct', 'timer_slack_ns')])},
5810            {'call': 'getrusage',
5811             'reason': set([('mm_struct', 'flags'),
5812                            ('task_struct', 'flags'),
5813                            ('task_struct', 'personality'),
5814                            ('task_struct', 'timer_slack_ns')])},
5815            {'call': 'sched_setscheduler',
5816             'reason': set([('task_struct', 'flags'),
5817                            ('task_struct', 'personality'),
5818                            ('task_struct', 'timer_slack_ns')])},
5819            {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
5820            {'call': 'setitimer',
5821             'reason': set([('task_struct', 'flags'),
5822                            ('task_struct', 'personality'),
5823                            ('task_struct', 'timer_slack_ns')])},
5824            {'call': 'ioprio_get',
5825             'reason': set([('task_struct', 'flags'),
5826                            ('task_struct', 'personality'),
5827                            ('task_struct', 'timer_slack_ns')])},
5828            {'call': 'vfork',
5829             'reason': set([('task_struct', 'flags'),
5830                            ('task_struct', 'personality'),
5831                            ('task_struct', 'timer_slack_ns')])},
5832            {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
5833            {'call': 'io_setup', 'reason': set([('mm_struct', 'flags')])},
5834            {'call': 'mremap', 'reason': set([('mm_struct', 'flags')])},
5835            {'call': 'io_destroy', 'reason': set([('mm_struct', 'flags')])},
5836            {'call': 'mbind', 'reason': set([('mm_struct', 'flags')])},
5837            {'call': 'move_pages',
5838             'reason': set([('mm_struct', 'flags'),
5839                            ('task_struct', 'flags'),
5840                            ('task_struct', 'personality'),
5841                            ('task_struct', 'timer_slack_ns')])},
5842            {'call': 'modify_ldt', 'reason': set([('mm_struct', 'flags')])},
5843            {'call': 'setpriority',
5844             'reason': set([('task_struct', 'flags'),
5845                            ('task_struct', 'personality'),
5846                            ('task_struct', 'timer_slack_ns')])},
5847            {'call': 'mincore', 'reason': set([('mm_struct', 'flags')])},
5848            {'call': 'clone',
5849             'reason': set([('task_struct', 'flags'),
5850                            ('task_struct', 'personality'),
5851                            ('task_struct', 'timer_slack_ns')])},
5852            {'call': 'msync', 'reason': set([('mm_struct', 'flags')])},
5853            {'call': 'sched_getparam',
5854             'reason': set([('task_struct', 'flags'),
5855                            ('task_struct', 'personality'),
5856                            ('task_struct', 'timer_slack_ns')])},
5857            {'call': 'io_cancel', 'reason': set([('mm_struct', 'flags')])}],
5858  'pread64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
5859              {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
5860              {'call': 'fadvise64_64',
5861               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
5862              {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
5863              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
5864              {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
5865              {'call': 'readahead', 'reason': set([('fd', 'flags')])},
5866              {'call': 'getdents', 'reason': set([('fd', 'flags')])},
5867              {'call': 'writev', 'reason': set([('fd', 'flags')])},
```



```
5868              {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
5869              {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
5870              {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
5871              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
5872              {'call': 'remap_file_pages',
5873               'reason': set([('file', 'f_mode')])},
5874              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
5875              {'call': 'read', 'reason': set([('fd', 'flags')])},
5876              {'call': 'fchown', 'reason': set([('fd', 'flags')])},
5877              {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
5878              {'call': 'utime', 'reason': set([('fd', 'flags')])},
5879              {'call': 'fsync', 'reason': set([('fd', 'flags')])},
5880              {'call': 'bpf', 'reason': set([('fd', 'flags')])},
5881              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
5882              {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
5883              {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
5884              {'call': 'sendto', 'reason': set([('fd', 'flags')])},
5885              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
5886              {'call': 'tee', 'reason': set([('fd', 'flags')])},
5887              {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
5888              {'call': 'lseek', 'reason': set([('fd', 'flags')])},
5889              {'call': 'connect', 'reason': set([('fd', 'flags')])},
5890              {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
5891              {'call': 'epoll_ctl',
5892               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
5893              {'call': 'flock',
5894               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
5895              {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
5896              {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
5897              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
5898              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
5899              {'call': 'accept4',
5900               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
5901              {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
5902              {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
5903              {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
5904              {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
5905              {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
5906              {'call': 'splice', 'reason': set([('fd', 'flags')])},
5907              {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
5908              {'call': 'preadv', 'reason': set([('fd', 'flags')])},
5909              {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
5910              {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
5911              {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
5912              {'call': 'socket', 'reason': set([('file', 'f_mode')])},
5913              {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
5914              {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
5915              {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
5916              {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
5917              {'call': 'perf_event_open',
5918               'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
5919              {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
5920              {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
5921              {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
5922              {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
5923              {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
5924              {'call': 'acct', 'reason': set([('file', 'f_mode')])},
5925              {'call': 'open', 'reason': set([('file', 'f_mode')])},
5926              {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
5927              {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
5928              {'call': 'dup', 'reason': set([('file', 'f_mode')])},
5929              {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
5930              {'call': 'setns', 'reason': set([('file', 'f_mode')])},
5931              {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
5932              {'call': 'listen', 'reason': set([('fd', 'flags')])},
5933              {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
```

```
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'msync', 'reason': set([('file', 'f_mode')])},
            {'call': 'open_by_handle_at',
             'reason': set([('file', 'f_mode')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'preadv': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
            {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
            {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
            {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
            {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
            {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
            {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
            {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
            {'call': 'flock', 'reason': set([('file', 'f_mode')])},
            {'call': 'openat', 'reason': set([('file', 'f_mode')])},
            {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
            {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
            {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
            {'call': 'socket', 'reason': set([('file', 'f_mode')])},
            {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
            {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
            {'call': 'open', 'reason': set([('file', 'f_mode')])},
            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'msync', 'reason': set([('file', 'f_mode')])},
            {'call': 'open_by_handle_at',
             'reason': set([('file', 'f_mode')])}],
 'preadv2': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
             {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
             {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
             {'call': 'remap_file_pages',
              'reason': set([('file', 'f_mode')])},
             {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
             {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
             {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
             {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
             {'call': 'flock', 'reason': set([('file', 'f_mode')])},
             {'call': 'openat', 'reason': set([('file', 'f_mode')])},
             {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
             {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
```



```
            {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
            {'call': 'socket', 'reason': set([('file', 'f_mode')])},
            {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
            {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
            {'call': 'acct', 'reason': set([('file', 'f_mode')])},
            {'call': 'open', 'reason': set([('file', 'f_mode')])},
            {'call': 'dup', 'reason': set([('file', 'f_mode')])},
            {'call': 'setns', 'reason': set([('file', 'f_mode')])},
            {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
            {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
            {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
            {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
            {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
            {'call': 'msync', 'reason': set([('file', 'f_mode')])},
            {'call': 'open_by_handle_at',
             'reason': set([('file', 'f_mode')])}],
 'preadv64': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
              {'call': 'remap_file_pages',
               'reason': set([('file', 'f_mode')])},
              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
              {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
              {'call': 'flock', 'reason': set([('file', 'f_mode')])},
              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
              {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
              {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
              {'call': 'socket', 'reason': set([('file', 'f_mode')])},
              {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
              {'call': 'perf_event_open',
               'reason': set([('file', 'f_mode')])},
              {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
              {'call': 'acct', 'reason': set([('file', 'f_mode')])},
              {'call': 'open', 'reason': set([('file', 'f_mode')])},
              {'call': 'dup', 'reason': set([('file', 'f_mode')])},
              {'call': 'setns', 'reason': set([('file', 'f_mode')])},
              {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
              {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
              {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
              {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
              {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
              {'call': 'msync', 'reason': set([('file', 'f_mode')])},
              {'call': 'open_by_handle_at',
               'reason': set([('file', 'f_mode')])}],
 'preadv64v2': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
                {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
                {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
                {'call': 'remap_file_pages',
                 'reason': set([('file', 'f_mode')])},
                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
                {'call': 'epoll_create1',
                 'reason': set([('file', 'f_mode')])},
                {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
                {'call': 'flock', 'reason': set([('file', 'f_mode')])},
                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
                {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
                {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
                {'call': 'perf_event_open',
```

```
                 'reason': set([('file', 'f_mode')])},
                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
                {'call': 'open', 'reason': set([('file', 'f_mode')])},
                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
                {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
                {'call': 'msync', 'reason': set([('file', 'f_mode')])},
                {'call': 'open_by_handle_at',
                 'reason': set([('file', 'f_mode')])}],
 'prlimit64': [{'call': 'setrlimit',
                'reason': set([('rlimit', 'rlim_cur'),
                               ('rlimit', 'rlim_max')])},
               {'call': 'old_getrlimit',
                'reason': set([('rlimit', 'rlim_cur'),
                               ('rlimit', 'rlim_max')])}],
 'pselect6': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'rt_sigtimedwait',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fadvise64_64',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'mq_unlink',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'swapoff',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'sched_getaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sched_setparam',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchmod',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'memfd_create',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'ioprio_set',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'readlinkat',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'io_getevents',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchown',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'mq_timedreceive',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'utime',
```



```
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'sched_setaffinity',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'semtimedop',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'settimeofday',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'sched_rr_get_interval',
               'reason': set([('mm_segment_t', 'seg'),
                              ('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'timerfd_gettime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'uselib',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'rt_sigprocmask',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sigaltstack',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'sched_setattr',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'migrate_pages',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'fchmodat',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'inotify_add_watch',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'timer_settime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'ftruncate',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'timer_gettime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'ioctl',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'perf_event_open',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'linkat',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'stime',
               'reason': set([('timespec', 'tv_nsec'),
                              ('timespec', 'tv_sec')])},
              {'call': 'rt_sigaction',
               'reason': set([('mm_segment_t', 'seg')])},
              {'call': 'futimesat',
               'reason': set([('timespec', 'tv_nsec'),
```

```
                                ('timespec', 'tv_sec')])},
             {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'poll',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'select',
              'reason': set([('compat_timespec', 'tv_nsec'),
                             ('compat_timespec', 'tv_sec'),
                             ('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'unlink',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'getpriority',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'nanosleep',
              'reason': set([('compat_timespec', 'tv_nsec'),
                             ('compat_timespec', 'tv_sec'),
                             ('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'mq_getsetattr',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'faccessat',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'get_robust_list',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'mq_timedsend',
              'reason': set([('mm_segment_t', 'seg'),
                             ('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'sched_getscheduler',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'swapon',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'epoll_wait',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'sched_getattr',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'fchownat',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'timerfd_settime',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'sched_setscheduler',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'ioprio_get',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'move_pages',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'setpriority',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'mq_notify',
              'reason': set([('timespec', 'tv_nsec'),
```



```
                                ('timespec', 'tv_sec')])},
             {'call': 'sendfile',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'clock_nanosleep',
              'reason': set([('compat_timespec', 'tv_nsec'),
                             ('compat_timespec', 'tv_sec'),
                             ('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'unlinkat',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'sched_getparam',
              'reason': set([('mm_segment_t', 'seg')])},
             {'call': 'futex',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'recvmmsg',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'sendfile64',
              'reason': set([('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])},
             {'call': 'ppoll',
              'reason': set([('compat_timespec', 'tv_nsec'),
                             ('compat_timespec', 'tv_sec'),
                             ('timespec', 'tv_nsec'),
                             ('timespec', 'tv_sec')])}],
 'ptrace': [{'call': 'keyctl',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'state')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'state')])},
            {'call': 'msgrcv',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'state')])},
            {'call': 'kill',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'state')])},
            {'call': 'pause', 'reason': set([('task_struct', 'state')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'state')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'state')])},
            {'call': 'ioprio_set',
             'reason': set([('task_struct', 'exit_state'),
                            ('task_struct', 'flags'),
                            ('task_struct', 'ptrace'),
                            ('task_struct', 'state')])},
            {'call': 'getppid',
```

```
6330              'reason': set([('task_struct', 'exit_state'),
6331                             ('task_struct', 'flags'),
6332                             ('task_struct', 'ptrace'),
6333                             ('task_struct', 'state')])},
6334             {'call': 'mq_timedreceive',
6335              'reason': set([('task_struct', 'exit_state'),
6336                             ('task_struct', 'flags'),
6337                             ('task_struct', 'ptrace'),
6338                             ('task_struct', 'state')])},
6339             {'call': 'capget',
6340              'reason': set([('task_struct', 'exit_state'),
6341                             ('task_struct', 'flags'),
6342                             ('task_struct', 'ptrace'),
6343                             ('task_struct', 'state')])},
6344             {'call': 'sched_setaffinity',
6345              'reason': set([('task_struct', 'exit_state'),
6346                             ('task_struct', 'flags'),
6347                             ('task_struct', 'ptrace'),
6348                             ('task_struct', 'state')])},
6349             {'call': 'signal',
6350              'reason': set([('task_struct', 'exit_state'),
6351                             ('task_struct', 'flags'),
6352                             ('task_struct', 'ptrace'),
6353                             ('task_struct', 'state')])},
6354             {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
6355             {'call': 'semtimedop',
6356              'reason': set([('task_struct', 'exit_state'),
6357                             ('task_struct', 'flags'),
6358                             ('task_struct', 'ptrace'),
6359                             ('task_struct', 'state')])},
6360             {'call': 'umount',
6361              'reason': set([('task_struct', 'exit_state'),
6362                             ('task_struct', 'flags'),
6363                             ('task_struct', 'ptrace'),
6364                             ('task_struct', 'state')])},
6365             {'call': 'sched_rr_get_interval',
6366              'reason': set([('task_struct', 'exit_state'),
6367                             ('task_struct', 'flags'),
6368                             ('task_struct', 'ptrace'),
6369                             ('task_struct', 'state')])},
6370             {'call': 'rt_sigprocmask',
6371              'reason': set([('task_struct', 'exit_state'),
6372                             ('task_struct', 'flags'),
6373                             ('task_struct', 'ptrace'),
6374                             ('task_struct', 'state')])},
6375             {'call': 'setsid',
6376              'reason': set([('task_struct', 'exit_state'),
6377                             ('task_struct', 'flags'),
6378                             ('task_struct', 'ptrace'),
6379                             ('task_struct', 'state')])},
6380             {'call': 'sigaltstack',
6381              'reason': set([('task_struct', 'exit_state'),
6382                             ('task_struct', 'flags'),
6383                             ('task_struct', 'ptrace'),
6384                             ('task_struct', 'state')])},
6385             {'call': 'sched_setattr',
6386              'reason': set([('task_struct', 'exit_state'),
6387                             ('task_struct', 'flags'),
6388                             ('task_struct', 'ptrace'),
6389                             ('task_struct', 'state')])},
6390             {'call': 'migrate_pages',
6391              'reason': set([('task_struct', 'exit_state'),
6392                             ('task_struct', 'flags'),
6393                             ('task_struct', 'ptrace'),
6394                             ('task_struct', 'state')])},
6395             {'call': 'getitimer',
```



```
6396              'reason': set([('task_struct', 'exit_state'),
6397                             ('task_struct', 'flags'),
6398                             ('task_struct', 'ptrace'),
6399                             ('task_struct', 'state')])},
6400             {'call': 'setpgid',
6401              'reason': set([('task_struct', 'exit_state'),
6402                             ('task_struct', 'flags'),
6403                             ('task_struct', 'ptrace'),
6404                             ('task_struct', 'state')])},
6405             {'call': 'rt_sigsuspend',
6406              'reason': set([('task_struct', 'state')])},
6407             {'call': 'getsid',
6408              'reason': set([('task_struct', 'exit_state'),
6409                             ('task_struct', 'flags'),
6410                             ('task_struct', 'ptrace'),
6411                             ('task_struct', 'state')])},
6412             {'call': 'prlimit64',
6413              'reason': set([('task_struct', 'exit_state'),
6414                             ('task_struct', 'flags'),
6415                             ('task_struct', 'ptrace'),
6416                             ('task_struct', 'state')])},
6417             {'call': 'perf_event_open',
6418              'reason': set([('task_struct', 'exit_state'),
6419                             ('task_struct', 'flags'),
6420                             ('task_struct', 'ptrace'),
6421                             ('task_struct', 'state')])},
6422             {'call': 'rt_sigaction',
6423              'reason': set([('task_struct', 'exit_state'),
6424                             ('task_struct', 'flags'),
6425                             ('task_struct', 'ptrace'),
6426                             ('task_struct', 'state')])},
6427             {'call': 'getpgid',
6428              'reason': set([('task_struct', 'exit_state'),
6429                             ('task_struct', 'flags'),
6430                             ('task_struct', 'ptrace'),
6431                             ('task_struct', 'state')])},
6432             {'call': 'getpriority',
6433              'reason': set([('task_struct', 'exit_state'),
6434                             ('task_struct', 'flags'),
6435                             ('task_struct', 'ptrace'),
6436                             ('task_struct', 'state')])},
6437             {'call': 'sigaction',
6438              'reason': set([('task_struct', 'exit_state'),
6439                             ('task_struct', 'flags'),
6440                             ('task_struct', 'ptrace'),
6441                             ('task_struct', 'state')])},
6442             {'call': 'setns',
6443              'reason': set([('task_struct', 'exit_state'),
6444                             ('task_struct', 'flags'),
6445                             ('task_struct', 'ptrace'),
6446                             ('task_struct', 'state')])},
6447             {'call': 'fork',
6448              'reason': set([('task_struct', 'exit_state'),
6449                             ('task_struct', 'flags'),
6450                             ('task_struct', 'ptrace'),
6451                             ('task_struct', 'state')])},
6452             {'call': 'get_robust_list',
6453              'reason': set([('task_struct', 'exit_state'),
6454                             ('task_struct', 'flags'),
6455                             ('task_struct', 'ptrace'),
6456                             ('task_struct', 'state')])},
6457             {'call': 'mq_timedsend',
6458              'reason': set([('task_struct', 'exit_state'),
6459                             ('task_struct', 'flags'),
6460                             ('task_struct', 'ptrace'),
6461                             ('task_struct', 'state')])},
```

```
6462             {'call': 'sched_getscheduler',
6463              'reason': set([('task_struct', 'exit_state'),
6464                             ('task_struct', 'flags'),
6465                             ('task_struct', 'ptrace'),
6466                             ('task_struct', 'state')])},
6467             {'call': 'epoll_wait', 'reason': set([('task_struct', 'state')])},
6468             {'call': 'sched_getattr',
6469              'reason': set([('task_struct', 'exit_state'),
6470                             ('task_struct', 'flags'),
6471                             ('task_struct', 'ptrace'),
6472                             ('task_struct', 'state')])},
6473             {'call': 'getrusage',
6474              'reason': set([('task_struct', 'exit_state'),
6475                             ('task_struct', 'flags'),
6476                             ('task_struct', 'ptrace'),
6477                             ('task_struct', 'state')])},
6478             {'call': 'sched_setscheduler',
6479              'reason': set([('task_struct', 'exit_state'),
6480                             ('task_struct', 'flags'),
6481                             ('task_struct', 'ptrace'),
6482                             ('task_struct', 'state')])},
6483             {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
6484             {'call': 'setitimer',
6485              'reason': set([('task_struct', 'exit_state'),
6486                             ('task_struct', 'flags'),
6487                             ('task_struct', 'ptrace'),
6488                             ('task_struct', 'state')])},
6489             {'call': 'ioprio_get',
6490              'reason': set([('task_struct', 'exit_state'),
6491                             ('task_struct', 'flags'),
6492                             ('task_struct', 'ptrace'),
6493                             ('task_struct', 'state')])},
6494             {'call': 'vfork',
6495              'reason': set([('task_struct', 'exit_state'),
6496                             ('task_struct', 'flags'),
6497                             ('task_struct', 'ptrace'),
6498                             ('task_struct', 'state')])},
6499             {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
6500             {'call': 'prctl',
6501              'reason': set([('task_struct', 'exit_state'),
6502                             ('task_struct', 'flags'),
6503                             ('task_struct', 'ptrace'),
6504                             ('task_struct', 'state')])},
6505             {'call': 'move_pages',
6506              'reason': set([('task_struct', 'exit_state'),
6507                             ('task_struct', 'flags'),
6508                             ('task_struct', 'ptrace'),
6509                             ('task_struct', 'state')])},
6510             {'call': 'setpriority',
6511              'reason': set([('task_struct', 'exit_state'),
6512                             ('task_struct', 'flags'),
6513                             ('task_struct', 'ptrace'),
6514                             ('task_struct', 'state')])},
6515             {'call': 'clone',
6516              'reason': set([('task_struct', 'exit_state'),
6517                             ('task_struct', 'flags'),
6518                             ('task_struct', 'ptrace'),
6519                             ('task_struct', 'state')])},
6520             {'call': 'sigsuspend', 'reason': set([('task_struct', 'state')])},
6521             {'call': 'sched_getparam',
6522              'reason': set([('task_struct', 'exit_state'),
6523                             ('task_struct', 'flags'),
6524                             ('task_struct', 'ptrace'),
6525                             ('task_struct', 'state')])}],
6526  'pwrite64': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
6527               {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
```



```
6528               {'call': 'fadvise64_64',
6529                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6530               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
6531               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
6532               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
6533               {'call': 'readahead', 'reason': set([('fd', 'flags')])},
6534               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
6535               {'call': 'writev', 'reason': set([('fd', 'flags')])},
6536               {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
6537               {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
6538               {'call': 'pread64', 'reason': set([('fd', 'flags')])},
6539               {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
6540               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
6541               {'call': 'remap_file_pages',
6542                'reason': set([('file', 'f_mode')])},
6543               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
6544               {'call': 'read', 'reason': set([('fd', 'flags')])},
6545               {'call': 'fchown', 'reason': set([('fd', 'flags')])},
6546               {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
6547               {'call': 'utime', 'reason': set([('fd', 'flags')])},
6548               {'call': 'fsync', 'reason': set([('fd', 'flags')])},
6549               {'call': 'bpf', 'reason': set([('fd', 'flags')])},
6550               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
6551               {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
6552               {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
6553               {'call': 'sendto', 'reason': set([('fd', 'flags')])},
6554               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
6555               {'call': 'tee', 'reason': set([('fd', 'flags')])},
6556               {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
6557               {'call': 'lseek', 'reason': set([('fd', 'flags')])},
6558               {'call': 'connect', 'reason': set([('fd', 'flags')])},
6559               {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
6560               {'call': 'epoll_ctl',
6561                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6562               {'call': 'flock',
6563                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6564               {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
6565               {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
6566               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
6567               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
6568               {'call': 'accept4',
6569                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6570               {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
6571               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
6572               {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
6573               {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
6574               {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
6575               {'call': 'splice', 'reason': set([('fd', 'flags')])},
6576               {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
6577               {'call': 'preadv', 'reason': set([('fd', 'flags')])},
6578               {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
6579               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
6580               {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
6581               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
6582               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
6583               {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
6584               {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
6585               {'call': 'perf_event_open',
6586                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6587               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
6588               {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
6589               {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
6590               {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
6591               {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
6592               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
6593               {'call': 'open', 'reason': set([('file', 'f_mode')])},
```

```
6594                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
6595                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
6596                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
6597                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
6598                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
6599                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
6600                {'call': 'listen', 'reason': set([('fd', 'flags')])},
6601                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
6602                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
6603                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
6604                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
6605                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
6606                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
6607                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
6608                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
6609                {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
6610                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
6611                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
6612                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
6613                {'call': 'readv', 'reason': set([('fd', 'flags')])},
6614                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
6615                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
6616                {'call': 'write', 'reason': set([('fd', 'flags')])},
6617                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
6618                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
6619                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
6620                {'call': 'msync', 'reason': set([('file', 'f_mode')])},
6621                {'call': 'open_by_handle_at',
6622                 'reason': set([('file', 'f_mode')])},
6623                {'call': 'bind', 'reason': set([('fd', 'flags')])},
6624                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
6625                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
6626  'pwritev': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
6627              {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
6628              {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
6629              {'call': 'remap_file_pages',
6630               'reason': set([('file', 'f_mode')])},
6631              {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
6632              {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
6633              {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
6634              {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
6635              {'call': 'flock', 'reason': set([('file', 'f_mode')])},
6636              {'call': 'openat', 'reason': set([('file', 'f_mode')])},
6637              {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
6638              {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
6639              {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
6640              {'call': 'socket', 'reason': set([('file', 'f_mode')])},
6641              {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
6642              {'call': 'perf_event_open', 'reason': set([('file', 'f_mode')])},
6643              {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
6644              {'call': 'acct', 'reason': set([('file', 'f_mode')])},
6645              {'call': 'open', 'reason': set([('file', 'f_mode')])},
6646              {'call': 'dup', 'reason': set([('file', 'f_mode')])},
6647              {'call': 'setns', 'reason': set([('file', 'f_mode')])},
6648              {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
6649              {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
6650              {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
6651              {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
6652              {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
6653              {'call': 'msync', 'reason': set([('file', 'f_mode')])},
6654              {'call': 'open_by_handle_at',
6655               'reason': set([('file', 'f_mode')])}],
6656  'pwritev2': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
6657               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
6658               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
6659               {'call': 'remap_file_pages',
```



```
6660                'reason': set([('file', 'f_mode')])},
6661               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
6662               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
6663               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
6664               {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
6665               {'call': 'flock', 'reason': set([('file', 'f_mode')])},
6666               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
6667               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
6668               {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
6669               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
6670               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
6671               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
6672               {'call': 'perf_event_open',
6673                'reason': set([('file', 'f_mode')])},
6674               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
6675               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
6676               {'call': 'open', 'reason': set([('file', 'f_mode')])},
6677               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
6678               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
6679               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
6680               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
6681               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
6682               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
6683               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
6684               {'call': 'msync', 'reason': set([('file', 'f_mode')])},
6685               {'call': 'open_by_handle_at',
6686                'reason': set([('file', 'f_mode')])}],
6687  'pwritev64': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
6688                {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
6689                {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
6690                {'call': 'remap_file_pages',
6691                 'reason': set([('file', 'f_mode')])},
6692                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
6693                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
6694                {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
6695                {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
6696                {'call': 'flock', 'reason': set([('file', 'f_mode')])},
6697                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
6698                {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
6699                {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
6700                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
6701                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
6702                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
6703                {'call': 'perf_event_open',
6704                 'reason': set([('file', 'f_mode')])},
6705                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
6706                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
6707                {'call': 'open', 'reason': set([('file', 'f_mode')])},
6708                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
6709                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
6710                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
6711                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
6712                {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
6713                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
6714                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
6715                {'call': 'msync', 'reason': set([('file', 'f_mode')])},
6716                {'call': 'open_by_handle_at',
6717                 'reason': set([('file', 'f_mode')])}],
6718  'pwritev64v2': [{'call': 'fadvise64_64',
6719                   'reason': set([('file', 'f_mode')])},
6720                  {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
6721                  {'call': 'memfd_create',
6722                   'reason': set([('file', 'f_mode')])},
6723                  {'call': 'remap_file_pages',
6724                   'reason': set([('file', 'f_mode')])},
6725                  {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
```

```
6726                 {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
6727                 {'call': 'epoll_create1',
6728                  'reason': set([('file', 'f_mode')])},
6729                 {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
6730                 {'call': 'flock', 'reason': set([('file', 'f_mode')])},
6731                 {'call': 'openat', 'reason': set([('file', 'f_mode')])},
6732                 {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
6733                 {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
6734                 {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
6735                 {'call': 'socket', 'reason': set([('file', 'f_mode')])},
6736                 {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
6737                 {'call': 'perf_event_open',
6738                  'reason': set([('file', 'f_mode')])},
6739                 {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
6740                 {'call': 'acct', 'reason': set([('file', 'f_mode')])},
6741                 {'call': 'open', 'reason': set([('file', 'f_mode')])},
6742                 {'call': 'dup', 'reason': set([('file', 'f_mode')])},
6743                 {'call': 'setns', 'reason': set([('file', 'f_mode')])},
6744                 {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
6745                 {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
6746                 {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
6747                 {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
6748                 {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
6749                 {'call': 'msync', 'reason': set([('file', 'f_mode')])},
6750                 {'call': 'open_by_handle_at',
6751                  'reason': set([('file', 'f_mode')])}],
6752  'quotactl': [{'call': 'syncfs',
6753                 'reason': set([('super_block', 's_flags'),
6754                                ('super_block', 's_quota_types')])},
6755                {'call': 'sysfs', 'reason': set([('filename', 'name')])},
6756                {'call': 'fadvise64_64',
6757                 'reason': set([('super_block', 's_flags'),
6758                                ('super_block', 's_quota_types')])},
6759                {'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
6760                {'call': 'swapoff', 'reason': set([('filename', 'name')])},
6761                {'call': 'ustat',
6762                 'reason': set([('super_block', 's_flags'),
6763                                ('super_block', 's_quota_types')])},
6764                {'call': 'umount',
6765                 'reason': set([('super_block', 's_flags'),
6766                                ('super_block', 's_quota_types')])},
6767                {'call': 'openat', 'reason': set([('filename', 'name')])},
6768                {'call': 'uselib', 'reason': set([('filename', 'name')])},
6769                {'call': 'renameat2', 'reason': set([('filename', 'name')])},
6770                {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
6771                {'call': 'acct', 'reason': set([('filename', 'name')])},
6772                {'call': 'open', 'reason': set([('filename', 'name')])},
6773                {'call': 'unlink', 'reason': set([('filename', 'name')])},
6774                {'call': 'rmdir', 'reason': set([('filename', 'name')])},
6775                {'call': 'swapon',
6776                 'reason': set([('filename', 'name'),
6777                                ('super_block', 's_flags'),
6778                                ('super_block', 's_quota_types')])},
6779                {'call': 'mq_open', 'reason': set([('filename', 'name')])},
6780                {'call': 'unlinkat', 'reason': set([('filename', 'name')])}],
6781  'read': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
6782           {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
6783           {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
6784           {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
6785           {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
6786           {'call': 'readahead', 'reason': set([('fd', 'flags')])},
6787           {'call': 'getdents', 'reason': set([('fd', 'flags')])},
6788           {'call': 'writev', 'reason': set([('fd', 'flags')])},
6789           {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
6790           {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
6791           {'call': 'pread64', 'reason': set([('fd', 'flags')])},
```



```
6792           {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
6793           {'call': 'fchown', 'reason': set([('fd', 'flags')])},
6794           {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
6795           {'call': 'utime', 'reason': set([('fd', 'flags')])},
6796           {'call': 'fsync', 'reason': set([('fd', 'flags')])},
6797           {'call': 'bpf', 'reason': set([('fd', 'flags')])},
6798           {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
6799           {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
6800           {'call': 'sendto', 'reason': set([('fd', 'flags')])},
6801           {'call': 'tee', 'reason': set([('fd', 'flags')])},
6802           {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
6803           {'call': 'lseek', 'reason': set([('fd', 'flags')])},
6804           {'call': 'connect', 'reason': set([('fd', 'flags')])},
6805           {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
6806           {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
6807           {'call': 'flock', 'reason': set([('fd', 'flags')])},
6808           {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
6809           {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
6810           {'call': 'accept4', 'reason': set([('fd', 'flags')])},
6811           {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
6812           {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
6813           {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
6814           {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
6815           {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
6816           {'call': 'splice', 'reason': set([('fd', 'flags')])},
6817           {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
6818           {'call': 'preadv', 'reason': set([('fd', 'flags')])},
6819           {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
6820           {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
6821           {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
6822           {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
6823           {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
6824           {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
6825           {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
6826           {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
6827           {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
6828           {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
6829           {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
6830           {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
6831           {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
6832           {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
6833           {'call': 'listen', 'reason': set([('fd', 'flags')])},
6834           {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
6835           {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
6836           {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
6837           {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
6838           {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
6839           {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
6840           {'call': 'llseek', 'reason': set([('fd', 'flags')])},
6841           {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
6842           {'call': 'readv', 'reason': set([('fd', 'flags')])},
6843           {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
6844           {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
6845           {'call': 'write', 'reason': set([('fd', 'flags')])},
6846           {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
6847           {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
6848           {'call': 'bind', 'reason': set([('fd', 'flags')])},
6849           {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
6850           {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
6851  'readahead': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
6852                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
6853                {'call': 'fadvise64_64',
6854                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6855                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
6856                {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
6857                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
```

```
6858                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
6859                {'call': 'writev', 'reason': set([('fd', 'flags')])},
6860                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
6861                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
6862                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
6863                {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
6864                {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
6865                {'call': 'remap_file_pages',
6866                 'reason': set([('file', 'f_mode')])},
6867                {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
6868                {'call': 'read', 'reason': set([('fd', 'flags')])},
6869                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
6870                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
6871                {'call': 'utime', 'reason': set([('fd', 'flags')])},
6872                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
6873                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
6874                {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
6875                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
6876                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
6877                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
6878                {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
6879                {'call': 'tee', 'reason': set([('fd', 'flags')])},
6880                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
6881                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
6882                {'call': 'connect', 'reason': set([('fd', 'flags')])},
6883                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
6884                {'call': 'epoll_ctl',
6885                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6886                {'call': 'flock',
6887                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6888                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
6889                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
6890                {'call': 'openat', 'reason': set([('file', 'f_mode')])},
6891                {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
6892                {'call': 'accept4',
6893                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6894                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
6895                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
6896                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
6897                {'call': 'inotify_add_watch',
6898                 'reason': set([('fd', 'flags')])},
6899                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
6900                {'call': 'splice', 'reason': set([('fd', 'flags')])},
6901                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
6902                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
6903                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
6904                {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
6905                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
6906                {'call': 'socket', 'reason': set([('file', 'f_mode')])},
6907                {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
6908                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
6909                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
6910                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
6911                {'call': 'perf_event_open',
6912                 'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
6913                {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
6914                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
6915                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
6916                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
6917                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
6918                {'call': 'acct', 'reason': set([('file', 'f_mode')])},
6919                {'call': 'open', 'reason': set([('file', 'f_mode')])},
6920                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
6921                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
6922                {'call': 'dup', 'reason': set([('file', 'f_mode')])},
6923                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
```



```
6924                {'call': 'setns', 'reason': set([('file', 'f_mode')])},
6925                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
6926                {'call': 'listen', 'reason': set([('fd', 'flags')])},
6927                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
6928                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
6929                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
6930                {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
6931                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
6932                {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
6933                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
6934                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
6935                {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
6936                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
6937                {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
6938                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
6939                {'call': 'readv', 'reason': set([('fd', 'flags')])},
6940                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
6941                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
6942                {'call': 'write', 'reason': set([('fd', 'flags')])},
6943                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
6944                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
6945                {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
6946                {'call': 'msync', 'reason': set([('file', 'f_mode')])},
6947                {'call': 'open_by_handle_at',
6948                 'reason': set([('file', 'f_mode')])},
6949                {'call': 'bind', 'reason': set([('fd', 'flags')])},
6950                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
6951                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
6952  'recvfrom': [{'call': 'swapoff', 'reason': set([('file', 'f_flags')])},
6953               {'call': 'memfd_create', 'reason': set([('file', 'f_flags')])},
6954               {'call': 'remap_file_pages',
6955                'reason': set([('file', 'f_flags')])},
6956               {'call': 'dup3', 'reason': set([('file', 'f_flags')])},
6957               {'call': 'socketpair', 'reason': set([('file', 'f_flags')])},
6958               {'call': 'epoll_create1', 'reason': set([('file', 'f_flags')])},
6959               {'call': 'epoll_ctl', 'reason': set([('file', 'f_flags')])},
6960               {'call': 'flock', 'reason': set([('file', 'f_flags')])},
6961               {'call': 'openat', 'reason': set([('file', 'f_flags')])},
6962               {'call': 'uselib', 'reason': set([('file', 'f_flags')])},
6963               {'call': 'accept4', 'reason': set([('file', 'f_flags')])},
6964               {'call': 'shmat', 'reason': set([('file', 'f_flags')])},
6965               {'call': 'socket', 'reason': set([('file', 'f_flags')])},
6966               {'call': 'pipe2', 'reason': set([('file', 'f_flags')])},
6967               {'call': 'perf_event_open',
6968                'reason': set([('file', 'f_flags')])},
6969               {'call': 'shmdt', 'reason': set([('file', 'f_flags')])},
6970               {'call': 'acct', 'reason': set([('file', 'f_flags')])},
6971               {'call': 'open', 'reason': set([('file', 'f_flags')])},
6972               {'call': 'mq_getsetattr', 'reason': set([('file', 'f_flags')])},
6973               {'call': 'dup', 'reason': set([('file', 'f_flags')])},
6974               {'call': 'setns', 'reason': set([('file', 'f_flags')])},
6975               {'call': 'shmctl', 'reason': set([('file', 'f_flags')])},
6976               {'call': 'swapon', 'reason': set([('file', 'f_flags')])},
6977               {'call': 'eventfd2', 'reason': set([('file', 'f_flags')])},
6978               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_flags')])},
6979               {'call': 'mq_open', 'reason': set([('file', 'f_flags')])},
6980               {'call': 'msync', 'reason': set([('file', 'f_flags')])},
6981               {'call': 'open_by_handle_at',
6982                'reason': set([('file', 'f_flags')])}],
6983  'recvmmsg': [{'call': 'rt_sigtimedwait',
6984                'reason': set([('timespec', 'tv_nsec'),
6985                               ('timespec', 'tv_sec')])},
6986               {'call': 'fadvise64_64',
6987                'reason': set([('timespec', 'tv_nsec'),
6988                               ('timespec', 'tv_sec')])},
6989               {'call': 'mq_unlink',
```

```
6990                 'reason': set([('timespec', 'tv_nsec'),
6991                                ('timespec', 'tv_sec')])},
6992                {'call': 'swapoff',
6993                 'reason': set([('timespec', 'tv_nsec'),
6994                                ('timespec', 'tv_sec')])},
6995                {'call': 'fchmod',
6996                 'reason': set([('timespec', 'tv_nsec'),
6997                                ('timespec', 'tv_sec')])},
6998                {'call': 'memfd_create',
6999                 'reason': set([('timespec', 'tv_nsec'),
7000                                ('timespec', 'tv_sec')])},
7001                {'call': 'readlinkat',
7002                 'reason': set([('timespec', 'tv_nsec'),
7003                                ('timespec', 'tv_sec')])},
7004                {'call': 'io_getevents',
7005                 'reason': set([('timespec', 'tv_nsec'),
7006                                ('timespec', 'tv_sec')])},
7007                {'call': 'fchown',
7008                 'reason': set([('timespec', 'tv_nsec'),
7009                                ('timespec', 'tv_sec')])},
7010                {'call': 'mq_timedreceive',
7011                 'reason': set([('timespec', 'tv_nsec'),
7012                                ('timespec', 'tv_sec')])},
7013                {'call': 'utime',
7014                 'reason': set([('timespec', 'tv_nsec'),
7015                                ('timespec', 'tv_sec')])},
7016                {'call': 'semtimedop',
7017                 'reason': set([('timespec', 'tv_nsec'),
7018                                ('timespec', 'tv_sec')])},
7019                {'call': 'recvfrom', 'reason': set([('msghdr', 'msg_flags')])},
7020                {'call': 'settimeofday',
7021                 'reason': set([('timespec', 'tv_nsec'),
7022                                ('timespec', 'tv_sec')])},
7023                {'call': 'sendto', 'reason': set([('msghdr', 'msg_flags')])},
7024                {'call': 'sched_rr_get_interval',
7025                 'reason': set([('timespec', 'tv_nsec'),
7026                                ('timespec', 'tv_sec')])},
7027                {'call': 'timerfd_gettime',
7028                 'reason': set([('timespec', 'tv_nsec'),
7029                                ('timespec', 'tv_sec')])},
7030                {'call': 'pselect6',
7031                 'reason': set([('timespec', 'tv_nsec'),
7032                                ('timespec', 'tv_sec')])},
7033                {'call': 'uselib',
7034                 'reason': set([('timespec', 'tv_nsec'),
7035                                ('timespec', 'tv_sec')])},
7036                {'call': 'fchmodat',
7037                 'reason': set([('timespec', 'tv_nsec'),
7038                                ('timespec', 'tv_sec')])},
7039                {'call': 'inotify_add_watch',
7040                 'reason': set([('timespec', 'tv_nsec'),
7041                                ('timespec', 'tv_sec')])},
7042                {'call': 'timer_settime',
7043                 'reason': set([('timespec', 'tv_nsec'),
7044                                ('timespec', 'tv_sec')])},
7045                {'call': 'ftruncate',
7046                 'reason': set([('timespec', 'tv_nsec'),
7047                                ('timespec', 'tv_sec')])},
7048                {'call': 'timer_gettime',
7049                 'reason': set([('timespec', 'tv_nsec'),
7050                                ('timespec', 'tv_sec')])},
7051                {'call': 'ioctl',
7052                 'reason': set([('timespec', 'tv_nsec'),
7053                                ('timespec', 'tv_sec')])},
7054                {'call': 'linkat',
7055                 'reason': set([('timespec', 'tv_nsec'),
```



```
7056                                ('timespec', 'tv_sec')])},
7057                {'call': 'stime',
7058                 'reason': set([('timespec', 'tv_nsec'),
7059                                ('timespec', 'tv_sec')])},
7060                {'call': 'futimesat',
7061                 'reason': set([('timespec', 'tv_nsec'),
7062                                ('timespec', 'tv_sec')])},
7063                {'call': 'poll',
7064                 'reason': set([('timespec', 'tv_nsec'),
7065                                ('timespec', 'tv_sec')])},
7066                {'call': 'select',
7067                 'reason': set([('timespec', 'tv_nsec'),
7068                                ('timespec', 'tv_sec')])},
7069                {'call': 'unlink',
7070                 'reason': set([('timespec', 'tv_nsec'),
7071                                ('timespec', 'tv_sec')])},
7072                {'call': 'nanosleep',
7073                 'reason': set([('timespec', 'tv_nsec'),
7074                                ('timespec', 'tv_sec')])},
7075                {'call': 'mq_getsetattr',
7076                 'reason': set([('timespec', 'tv_nsec'),
7077                                ('timespec', 'tv_sec')])},
7078                {'call': 'faccessat',
7079                 'reason': set([('timespec', 'tv_nsec'),
7080                                ('timespec', 'tv_sec')])},
7081                {'call': 'mq_timedsend',
7082                 'reason': set([('timespec', 'tv_nsec'),
7083                                ('timespec', 'tv_sec')])},
7084                {'call': 'swapon',
7085                 'reason': set([('timespec', 'tv_nsec'),
7086                                ('timespec', 'tv_sec')])},
7087                {'call': 'epoll_wait',
7088                 'reason': set([('timespec', 'tv_nsec'),
7089                                ('timespec', 'tv_sec')])},
7090                {'call': 'fchownat',
7091                 'reason': set([('timespec', 'tv_nsec'),
7092                                ('timespec', 'tv_sec')])},
7093                {'call': 'timerfd_settime',
7094                 'reason': set([('timespec', 'tv_nsec'),
7095                                ('timespec', 'tv_sec')])},
7096                {'call': 'mq_notify',
7097                 'reason': set([('timespec', 'tv_nsec'),
7098                                ('timespec', 'tv_sec')])},
7099                {'call': 'sendfile',
7100                 'reason': set([('timespec', 'tv_nsec'),
7101                                ('timespec', 'tv_sec')])},
7102                {'call': 'clock_nanosleep',
7103                 'reason': set([('timespec', 'tv_nsec'),
7104                                ('timespec', 'tv_sec')])},
7105                {'call': 'unlinkat',
7106                 'reason': set([('timespec', 'tv_nsec'),
7107                                ('timespec', 'tv_sec')])},
7108                {'call': 'futex',
7109                 'reason': set([('timespec', 'tv_nsec'),
7110                                ('timespec', 'tv_sec')])},
7111                {'call': 'sendfile64',
7112                 'reason': set([('timespec', 'tv_nsec'),
7113                                ('timespec', 'tv_sec')])},
7114                {'call': 'ppoll',
7115                 'reason': set([('timespec', 'tv_nsec'),
7116                                ('timespec', 'tv_sec')])}],
7117  'remap_file_pages': [{'call': 'shmdt',
7118                        'reason': set([('vm_area_struct', 'vm_end'),
7119                                       ('vm_area_struct', 'vm_flags'),
7120                                       ('vm_area_struct', 'vm_start')])},
7121                       {'call': 'brk',
```

```
7122                         'reason': set([('vm_area_struct', 'vm_end'),
7123                                        ('vm_area_struct', 'vm_flags'),
7124                                        ('vm_area_struct', 'vm_start')])},
7125                        {'call': 'get_mempolicy',
7126                         'reason': set([('vm_area_struct', 'vm_end'),
7127                                        ('vm_area_struct', 'vm_flags'),
7128                                        ('vm_area_struct', 'vm_start')])},
7129                        {'call': 'munlockall',
7130                         'reason': set([('vm_area_struct', 'vm_end'),
7131                                        ('vm_area_struct', 'vm_flags'),
7132                                        ('vm_area_struct', 'vm_start')])},
7133                        {'call': 'pkey_mprotect',
7134                         'reason': set([('vm_area_struct', 'vm_end'),
7135                                        ('vm_area_struct', 'vm_flags'),
7136                                        ('vm_area_struct', 'vm_start')])},
7137                        {'call': 'madvise',
7138                         'reason': set([('vm_area_struct', 'vm_end'),
7139                                        ('vm_area_struct', 'vm_flags'),
7140                                        ('vm_area_struct', 'vm_start')])},
7141                        {'call': 'mprotect',
7142                         'reason': set([('vm_area_struct', 'vm_end'),
7143                                        ('vm_area_struct', 'vm_flags'),
7144                                        ('vm_area_struct', 'vm_start')])},
7145                        {'call': 'mremap',
7146                         'reason': set([('vm_area_struct', 'vm_end'),
7147                                        ('vm_area_struct', 'vm_flags'),
7148                                        ('vm_area_struct', 'vm_start')])},
7149                        {'call': 'prctl',
7150                         'reason': set([('vm_area_struct', 'vm_end'),
7151                                        ('vm_area_struct', 'vm_flags'),
7152                                        ('vm_area_struct', 'vm_start')])},
7153                        {'call': 'munlock',
7154                         'reason': set([('vm_area_struct', 'vm_end'),
7155                                        ('vm_area_struct', 'vm_flags'),
7156                                        ('vm_area_struct', 'vm_start')])},
7157                        {'call': 'mincore',
7158                         'reason': set([('vm_area_struct', 'vm_end'),
7159                                        ('vm_area_struct', 'vm_flags'),
7160                                        ('vm_area_struct', 'vm_start')])},
7161                        {'call': 'msync',
7162                         'reason': set([('vm_area_struct', 'vm_end'),
7163                                        ('vm_area_struct', 'vm_flags'),
7164                                        ('vm_area_struct', 'vm_start')])},
7165                        {'call': 'mlockall',
7166                         'reason': set([('vm_area_struct', 'vm_end'),
7167                                        ('vm_area_struct', 'vm_flags'),
7168                                        ('vm_area_struct', 'vm_start')])}],
7169      'renameat2': [{'call': 'sysfs',
7170                     'reason': set([('filename', 'name'),
7171                                    ('filename', 'refcnt')])},
7172                    {'call': 'mq_unlink',
7173                     'reason': set([('filename', 'name'),
7174                                    ('filename', 'refcnt')])},
7175                    {'call': 'swapoff',
7176                     'reason': set([('filename', 'name'),
7177                                    ('filename', 'refcnt')])},
7178                    {'call': 'openat',
7179                     'reason': set([('filename', 'name'),
7180                                    ('filename', 'refcnt')])},
7181                    {'call': 'uselib',
7182                     'reason': set([('filename', 'name'),
7183                                    ('filename', 'refcnt')])},
7184                    {'call': 'symlinkat',
7185                     'reason': set([('filename', 'name'),
7186                                    ('filename', 'refcnt')])},
7187                    {'call': 'quotactl',
```



```
7188                     'reason': set([('filename', 'name'),
7189                                    ('filename', 'refcnt')])},
7190                    {'call': 'acct',
7191                     'reason': set([('filename', 'name'),
7192                                    ('filename', 'refcnt')])},
7193                    {'call': 'open',
7194                     'reason': set([('filename', 'name'),
7195                                    ('filename', 'refcnt')])},
7196                    {'call': 'unlink',
7197                     'reason': set([('filename', 'name'),
7198                                    ('filename', 'refcnt')])},
7199                    {'call': 'rmdir',
7200                     'reason': set([('filename', 'name'),
7201                                    ('filename', 'refcnt')])},
7202                    {'call': 'swapon',
7203                     'reason': set([('filename', 'name'),
7204                                    ('filename', 'refcnt')])},
7205                    {'call': 'mq_open',
7206                     'reason': set([('filename', 'name'),
7207                                    ('filename', 'refcnt')])},
7208                    {'call': 'unlinkat',
7209                     'reason': set([('filename', 'name'),
7210                                    ('filename', 'refcnt')])}],
7211      'rt_sigqueueinfo': [{'call': 'rt_sigtimedwait',
7212                           'reason': set([('siginfo', 'si_code')])},
7213                          {'call': 'kill',
7214                           'reason': set([('siginfo', 'si_code')])},
7215                          {'call': 'timer_create',
7216                           'reason': set([('siginfo', 'si_code')])},
7217                          {'call': 'tgkill',
7218                           'reason': set([('siginfo', 'si_code')])},
7219                          {'call': 'rt_tgsigqueueinfo',
7220                           'reason': set([('siginfo', 'si_code')])},
7221                          {'call': 'rt_sigreturn',
7222                           'reason': set([('siginfo', 'si_code')])},
7223                          {'call': 'tkill',
7224                           'reason': set([('siginfo', 'si_code')])}],
7225      'rt_sigreturn': [{'call': 'keyctl',
7226                        'reason': set([('mm_segment_t', 'seg'),
7227                                       ('thread_struct', 'uaccess_err')])},
7228                       {'call': 'rt_sigtimedwait',
7229                        'reason': set([('mm_segment_t', 'seg'),
7230                                       ('thread_struct', 'uaccess_err')])},
7231                       {'call': 'iopl',
7232                        'reason': set([('mm_segment_t', 'seg'),
7233                                       ('thread_struct', 'uaccess_err')])},
7234                       {'call': 'msgrcv',
7235                        'reason': set([('mm_segment_t', 'seg'),
7236                                       ('thread_struct', 'uaccess_err')])},
7237                       {'call': 'kill',
7238                        'reason': set([('mm_segment_t', 'seg'),
7239                                       ('thread_struct', 'uaccess_err')])},
7240                       {'call': 'sched_getaffinity',
7241                        'reason': set([('mm_segment_t', 'seg'),
7242                                       ('thread_struct', 'uaccess_err')])},
7243                       {'call': 'sched_setparam',
7244                        'reason': set([('mm_segment_t', 'seg'),
7245                                       ('thread_struct', 'uaccess_err')])},
7246                       {'call': 'ioprio_set',
7247                        'reason': set([('mm_segment_t', 'seg'),
7248                                       ('thread_struct', 'uaccess_err')])},
7249                       {'call': 'getppid',
7250                        'reason': set([('mm_segment_t', 'seg'),
7251                                       ('thread_struct', 'uaccess_err')])},
7252                       {'call': 'ioperm',
7253                        'reason': set([('mm_segment_t', 'seg'),
```

```
7254                                    ('thread_struct', 'uaccess_err')])},
7255                  {'call': 'mq_timedreceive',
7256                   'reason': set([('mm_segment_t', 'seg'),
7257                                  ('thread_struct', 'uaccess_err')])},
7258                  {'call': 'capget',
7259                   'reason': set([('mm_segment_t', 'seg'),
7260                                  ('thread_struct', 'uaccess_err')])},
7261                  {'call': 'sched_setaffinity',
7262                   'reason': set([('mm_segment_t', 'seg'),
7263                                  ('thread_struct', 'uaccess_err')])},
7264                  {'call': 'signal',
7265                   'reason': set([('mm_segment_t', 'seg'),
7266                                  ('thread_struct', 'uaccess_err')])},
7267                  {'call': 'semtimedop',
7268                   'reason': set([('mm_segment_t', 'seg'),
7269                                  ('thread_struct', 'uaccess_err')])},
7270                  {'call': 'umount',
7271                   'reason': set([('mm_segment_t', 'seg'),
7272                                  ('thread_struct', 'uaccess_err')])},
7273                  {'call': 'sched_rr_get_interval',
7274                   'reason': set([('mm_segment_t', 'seg'),
7275                                  ('thread_struct', 'uaccess_err')])},
7276                  {'call': 'rt_sigprocmask',
7277                   'reason': set([('mm_segment_t', 'seg'),
7278                                  ('thread_struct', 'uaccess_err')])},
7279                  {'call': 'setsid',
7280                   'reason': set([('mm_segment_t', 'seg'),
7281                                  ('thread_struct', 'uaccess_err')])},
7282                  {'call': 'sigaltstack',
7283                   'reason': set([('mm_segment_t', 'seg'),
7284                                  ('thread_struct', 'uaccess_err')])},
7285                  {'call': 'sched_setattr',
7286                   'reason': set([('mm_segment_t', 'seg'),
7287                                  ('thread_struct', 'uaccess_err')])},
7288                  {'call': 'migrate_pages',
7289                   'reason': set([('mm_segment_t', 'seg'),
7290                                  ('thread_struct', 'uaccess_err')])},
7291                  {'call': 'getitimer',
7292                   'reason': set([('mm_segment_t', 'seg'),
7293                                  ('thread_struct', 'uaccess_err')])},
7294                  {'call': 'setpgid',
7295                   'reason': set([('mm_segment_t', 'seg'),
7296                                  ('thread_struct', 'uaccess_err')])},
7297                  {'call': 'getsid',
7298                   'reason': set([('mm_segment_t', 'seg'),
7299                                  ('thread_struct', 'uaccess_err')])},
7300                  {'call': 'prlimit64',
7301                   'reason': set([('mm_segment_t', 'seg'),
7302                                  ('thread_struct', 'uaccess_err')])},
7303                  {'call': 'perf_event_open',
7304                   'reason': set([('mm_segment_t', 'seg'),
7305                                  ('thread_struct', 'uaccess_err')])},
7306                  {'call': 'rt_sigaction',
7307                   'reason': set([('mm_segment_t', 'seg'),
7308                                  ('thread_struct', 'uaccess_err')])},
7309                  {'call': 'getpgid',
7310                   'reason': set([('mm_segment_t', 'seg'),
7311                                  ('thread_struct', 'uaccess_err')])},
7312                  {'call': 'getpriority',
7313                   'reason': set([('mm_segment_t', 'seg'),
7314                                  ('thread_struct', 'uaccess_err')])},
7315                  {'call': 'sigaction',
7316                   'reason': set([('mm_segment_t', 'seg'),
7317                                  ('thread_struct', 'uaccess_err')])},
7318                  {'call': 'setns',
7319                   'reason': set([('mm_segment_t', 'seg'),
```



```
7320                                    ('thread_struct', 'uaccess_err')])},
7321                  {'call': 'fork',
7322                   'reason': set([('mm_segment_t', 'seg'),
7323                                  ('thread_struct', 'uaccess_err')])},
7324                  {'call': 'get_robust_list',
7325                   'reason': set([('mm_segment_t', 'seg'),
7326                                  ('thread_struct', 'uaccess_err')])},
7327                  {'call': 'mq_timedsend',
7328                   'reason': set([('mm_segment_t', 'seg'),
7329                                  ('thread_struct', 'uaccess_err')])},
7330                  {'call': 'sched_getscheduler',
7331                   'reason': set([('mm_segment_t', 'seg'),
7332                                  ('thread_struct', 'uaccess_err')])},
7333                  {'call': 'ptrace',
7334                   'reason': set([('mm_segment_t', 'seg'),
7335                                  ('thread_struct', 'uaccess_err')])},
7336                  {'call': 'sched_getattr',
7337                   'reason': set([('mm_segment_t', 'seg'),
7338                                  ('thread_struct', 'uaccess_err')])},
7339                  {'call': 'getrusage',
7340                   'reason': set([('mm_segment_t', 'seg'),
7341                                  ('thread_struct', 'uaccess_err')])},
7342                  {'call': 'sched_setscheduler',
7343                   'reason': set([('mm_segment_t', 'seg'),
7344                                  ('thread_struct', 'uaccess_err')])},
7345                  {'call': 'setitimer',
7346                   'reason': set([('mm_segment_t', 'seg'),
7347                                  ('thread_struct', 'uaccess_err')])},
7348                  {'call': 'ioprio_get',
7349                   'reason': set([('mm_segment_t', 'seg'),
7350                                  ('thread_struct', 'uaccess_err')])},
7351                  {'call': 'vfork',
7352                   'reason': set([('mm_segment_t', 'seg'),
7353                                  ('thread_struct', 'uaccess_err')])},
7354                  {'call': 'prctl',
7355                   'reason': set([('mm_segment_t', 'seg'),
7356                                  ('thread_struct', 'uaccess_err')])},
7357                  {'call': 'move_pages',
7358                   'reason': set([('mm_segment_t', 'seg'),
7359                                  ('thread_struct', 'uaccess_err')])},
7360                  {'call': 'setpriority',
7361                   'reason': set([('mm_segment_t', 'seg'),
7362                                  ('thread_struct', 'uaccess_err')])},
7363                  {'call': 'clone',
7364                   'reason': set([('mm_segment_t', 'seg'),
7365                                  ('thread_struct', 'uaccess_err')])},
7366                  {'call': 'sched_getparam',
7367                   'reason': set([('mm_segment_t', 'seg'),
7368                                  ('thread_struct', 'uaccess_err')])}],
7369  'rt_sigtimedwait': [{'call': 'keyctl',
7370                       'reason': set([('mm_segment_t', 'seg'),
7371                                      ('task_struct', 'timer_slack_ns')])},
7372                      {'call': 'iopl',
7373                       'reason': set([('mm_segment_t', 'seg')])},
7374                      {'call': 'msgrcv',
7375                       'reason': set([('mm_segment_t', 'seg'),
7376                                      ('task_struct', 'timer_slack_ns')])},
7377                      {'call': 'kill',
7378                       'reason': set([('mm_segment_t', 'seg'),
7379                                      ('siginfo', 'si_code'),
7380                                      ('siginfo', 'si_signo'),
7381                                      ('task_struct', 'timer_slack_ns')])},
7382                      {'call': 'sched_getaffinity',
7383                       'reason': set([('mm_segment_t', 'seg'),
7384                                      ('task_struct', 'timer_slack_ns')])},
7385                      {'call': 'sched_setparam',
```

```
7386                     'reason': set([('mm_segment_t', 'seg'),
7387                                    ('task_struct', 'timer_slack_ns')])},
7388                  {'call': 'ioprio_set',
7389                   'reason': set([('mm_segment_t', 'seg'),
7390                                  ('task_struct', 'timer_slack_ns')])},
7391                  {'call': 'getppid',
7392                   'reason': set([('mm_segment_t', 'seg'),
7393                                  ('task_struct', 'timer_slack_ns')])},
7394                  {'call': 'ioperm',
7395                   'reason': set([('mm_segment_t', 'seg')])},
7396                  {'call': 'mq_timedreceive',
7397                   'reason': set([('mm_segment_t', 'seg'),
7398                                  ('task_struct', 'timer_slack_ns')])},
7399                  {'call': 'capget',
7400                   'reason': set([('mm_segment_t', 'seg'),
7401                                  ('task_struct', 'timer_slack_ns')])},
7402                  {'call': 'sched_setaffinity',
7403                   'reason': set([('mm_segment_t', 'seg'),
7404                                  ('task_struct', 'timer_slack_ns')])},
7405                  {'call': 'signal',
7406                   'reason': set([('mm_segment_t', 'seg'),
7407                                  ('task_struct', 'timer_slack_ns')])},
7408                  {'call': 'semtimedop',
7409                   'reason': set([('mm_segment_t', 'seg'),
7410                                  ('task_struct', 'timer_slack_ns')])},
7411                  {'call': 'umount',
7412                   'reason': set([('mm_segment_t', 'seg'),
7413                                  ('task_struct', 'timer_slack_ns')])},
7414                  {'call': 'timer_create',
7415                   'reason': set([('siginfo', 'si_code'),
7416                                  ('siginfo', 'si_signo')])},
7417                  {'call': 'sched_rr_get_interval',
7418                   'reason': set([('mm_segment_t', 'seg'),
7419                                  ('task_struct', 'timer_slack_ns')])},
7420                  {'call': 'rt_sigqueueinfo',
7421                   'reason': set([('siginfo', 'si_code'),
7422                                  ('siginfo', 'si_signo')])},
7423                  {'call': 'tgkill',
7424                   'reason': set([('siginfo', 'si_code'),
7425                                  ('siginfo', 'si_signo')])},
7426                  {'call': 'rt_sigprocmask',
7427                   'reason': set([('mm_segment_t', 'seg'),
7428                                  ('task_struct', 'timer_slack_ns')])},
7429                  {'call': 'setsid',
7430                   'reason': set([('mm_segment_t', 'seg'),
7431                                  ('task_struct', 'timer_slack_ns')])},
7432                  {'call': 'sigaltstack',
7433                   'reason': set([('mm_segment_t', 'seg'),
7434                                  ('task_struct', 'timer_slack_ns')])},
7435                  {'call': 'sched_setattr',
7436                   'reason': set([('mm_segment_t', 'seg'),
7437                                  ('task_struct', 'timer_slack_ns')])},
7438                  {'call': 'migrate_pages',
7439                   'reason': set([('mm_segment_t', 'seg'),
7440                                  ('task_struct', 'timer_slack_ns')])},
7441                  {'call': 'getitimer',
7442                   'reason': set([('mm_segment_t', 'seg'),
7443                                  ('task_struct', 'timer_slack_ns')])},
7444                  {'call': 'setpgid',
7445                   'reason': set([('mm_segment_t', 'seg'),
7446                                  ('task_struct', 'timer_slack_ns')])},
7447                  {'call': 'getsid',
7448                   'reason': set([('mm_segment_t', 'seg'),
7449                                  ('task_struct', 'timer_slack_ns')])},
7450                  {'call': 'prlimit64',
7451                   'reason': set([('mm_segment_t', 'seg'),
```



```
7452                                  ('task_struct', 'timer_slack_ns')])},
7453                  {'call': 'perf_event_open',
7454                   'reason': set([('mm_segment_t', 'seg'),
7455                                  ('task_struct', 'timer_slack_ns')])},
7456                  {'call': 'rt_sigaction',
7457                   'reason': set([('mm_segment_t', 'seg'),
7458                                  ('task_struct', 'timer_slack_ns')])},
7459                  {'call': 'getpgid',
7460                   'reason': set([('mm_segment_t', 'seg'),
7461                                  ('task_struct', 'timer_slack_ns')])},
7462                  {'call': 'getpriority',
7463                   'reason': set([('mm_segment_t', 'seg'),
7464                                  ('task_struct', 'timer_slack_ns')])},
7465                  {'call': 'sigaction',
7466                   'reason': set([('mm_segment_t', 'seg'),
7467                                  ('task_struct', 'timer_slack_ns')])},
7468                  {'call': 'rt_tgsigqueueinfo',
7469                   'reason': set([('siginfo', 'si_code'),
7470                                  ('siginfo', 'si_signo')])},
7471                  {'call': 'setns',
7472                   'reason': set([('mm_segment_t', 'seg'),
7473                                  ('task_struct', 'timer_slack_ns')])},
7474                  {'call': 'fork',
7475                   'reason': set([('mm_segment_t', 'seg'),
7476                                  ('task_struct', 'timer_slack_ns')])},
7477                  {'call': 'get_robust_list',
7478                   'reason': set([('mm_segment_t', 'seg'),
7479                                  ('task_struct', 'timer_slack_ns')])},
7480                  {'call': 'mq_timedsend',
7481                   'reason': set([('mm_segment_t', 'seg'),
7482                                  ('task_struct', 'timer_slack_ns')])},
7483                  {'call': 'sched_getscheduler',
7484                   'reason': set([('mm_segment_t', 'seg'),
7485                                  ('task_struct', 'timer_slack_ns')])},
7486                  {'call': 'ptrace',
7487                   'reason': set([('mm_segment_t', 'seg'),
7488                                  ('task_struct', 'timer_slack_ns')])},
7489                  {'call': 'sched_getattr',
7490                   'reason': set([('mm_segment_t', 'seg'),
7491                                  ('task_struct', 'timer_slack_ns')])},
7492                  {'call': 'getrusage',
7493                   'reason': set([('mm_segment_t', 'seg'),
7494                                  ('task_struct', 'timer_slack_ns')])},
7495                  {'call': 'sched_setscheduler',
7496                   'reason': set([('mm_segment_t', 'seg'),
7497                                  ('task_struct', 'timer_slack_ns')])},
7498                  {'call': 'setitimer',
7499                   'reason': set([('mm_segment_t', 'seg'),
7500                                  ('task_struct', 'timer_slack_ns')])},
7501                  {'call': 'ioprio_get',
7502                   'reason': set([('mm_segment_t', 'seg'),
7503                                  ('task_struct', 'timer_slack_ns')])},
7504                  {'call': 'vfork',
7505                   'reason': set([('mm_segment_t', 'seg'),
7506                                  ('task_struct', 'timer_slack_ns')])},
7507                  {'call': 'prctl',
7508                   'reason': set([('mm_segment_t', 'seg'),
7509                                  ('task_struct', 'timer_slack_ns')])},
7510                  {'call': 'move_pages',
7511                   'reason': set([('mm_segment_t', 'seg'),
7512                                  ('task_struct', 'timer_slack_ns')])},
7513                  {'call': 'rt_sigreturn',
7514                   'reason': set([('siginfo', 'si_code'),
7515                                  ('siginfo', 'si_signo')])},
7516                  {'call': 'tkill',
7517                   'reason': set([('siginfo', 'si_code'),
```

```
                                    ('siginfo', 'si_signo')])},
                    {'call': 'setpriority',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'clone',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])},
                    {'call': 'sched_getparam',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'timer_slack_ns')])}],
 'rt_tgsigqueueinfo': [{'call': 'rt_sigtimedwait',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'kill',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'timer_create',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'rt_sigqueueinfo',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'tgkill',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'rt_sigreturn',
                        'reason': set([('siginfo', 'si_code')])},
                       {'call': 'tkill',
                        'reason': set([('siginfo', 'si_code')])}],
 'sched_getattr': [{'call': 'keyctl',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'msgrcv',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'kill',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_setparam',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'ioprio_set',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'getppid',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'ioperm',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'capget',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
```



```
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_setaffinity',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'signal',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'semtimedop',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'umount',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_rr_get_interval',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'rt_sigprocmask',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'setsid',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sigaltstack',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'sched_setattr',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('sched_attr', 'size'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'migrate_pages',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'getitimer',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'setpgid',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'getsid',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'prlimit64',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'perf_event_open',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
                   {'call': 'rt_sigaction',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('task_struct', 'policy'),
                                   ('task_struct', 'sched_reset_on_fork')])},
```

```
                    {'call': 'getpgid',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'getpriority',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'sigaction',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'setns',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'fork',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'get_robust_list',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'sched_getscheduler',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'ptrace',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'getrusage',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'sched_setscheduler',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'setitimer',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'ioprio_get',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'vfork',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'prctl',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'move_pages',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'setpriority',
                     'reason': set([('mm_segment_t', 'seg'),
```



```
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'clone',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])},
                    {'call': 'sched_getparam',
                     'reason': set([('mm_segment_t', 'seg'),
                                    ('task_struct', 'policy'),
                                    ('task_struct', 'sched_reset_on_fork')])}],
 'sched_getparam': [{'call': 'keyctl',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigtimedwait',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'msgrcv',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'kill',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_getaffinity',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setparam',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'ioprio_set',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getppid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'mq_timedreceive',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'capget',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setaffinity',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'signal',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'semtimedop',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'umount',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_rr_get_interval',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigprocmask',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setsid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sigaltstack',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'sched_setattr',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'migrate_pages',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getitimer',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'setpgid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getsid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'prlimit64',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'perf_event_open',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'rt_sigaction',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getpgid',
                     'reason': set([('task_struct', 'policy')])},
                    {'call': 'getpriority',
                     'reason': set([('task_struct', 'policy')])},
```

```
7782                     {'call': 'sigaction',
7783                      'reason': set([('task_struct', 'policy')])},
7784                     {'call': 'setns',
7785                      'reason': set([('task_struct', 'policy')])},
7786                     {'call': 'fork',
7787                      'reason': set([('task_struct', 'policy')])},
7788                     {'call': 'get_robust_list',
7789                      'reason': set([('task_struct', 'policy')])},
7790                     {'call': 'mq_timedsend',
7791                      'reason': set([('task_struct', 'policy')])},
7792                     {'call': 'sched_getscheduler',
7793                      'reason': set([('task_struct', 'policy')])},
7794                     {'call': 'ptrace',
7795                      'reason': set([('task_struct', 'policy')])},
7796                     {'call': 'sched_getattr',
7797                      'reason': set([('task_struct', 'policy')])},
7798                     {'call': 'getrusage',
7799                      'reason': set([('task_struct', 'policy')])},
7800                     {'call': 'sched_setscheduler',
7801                      'reason': set([('task_struct', 'policy')])},
7802                     {'call': 'setitimer',
7803                      'reason': set([('task_struct', 'policy')])},
7804                     {'call': 'ioprio_get',
7805                      'reason': set([('task_struct', 'policy')])},
7806                     {'call': 'vfork',
7807                      'reason': set([('task_struct', 'policy')])},
7808                     {'call': 'prctl',
7809                      'reason': set([('task_struct', 'policy')])},
7810                     {'call': 'move_pages',
7811                      'reason': set([('task_struct', 'policy')])},
7812                     {'call': 'setpriority',
7813                      'reason': set([('task_struct', 'policy')])},
7814                     {'call': 'clone',
7815                      'reason': set([('task_struct', 'policy')])}],
7816  'sched_getscheduler': [{'call': 'keyctl',
7817                          'reason': set([('task_struct',
7818                                          'sched_reset_on_fork')])},
7819                         {'call': 'rt_sigtimedwait',
7820                          'reason': set([('task_struct',
7821                                          'sched_reset_on_fork')])},
7822                         {'call': 'msgrcv',
7823                          'reason': set([('task_struct',
7824                                          'sched_reset_on_fork')])},
7825                         {'call': 'kill',
7826                          'reason': set([('task_struct',
7827                                          'sched_reset_on_fork')])},
7828                         {'call': 'sched_getaffinity',
7829                          'reason': set([('task_struct',
7830                                          'sched_reset_on_fork')])},
7831                         {'call': 'sched_setparam',
7832                          'reason': set([('task_struct',
7833                                          'sched_reset_on_fork')])},
7834                         {'call': 'ioprio_set',
7835                          'reason': set([('task_struct',
7836                                          'sched_reset_on_fork')])},
7837                         {'call': 'getppid',
7838                          'reason': set([('task_struct',
7839                                          'sched_reset_on_fork')])},
7840                         {'call': 'mq_timedreceive',
7841                          'reason': set([('task_struct',
7842                                          'sched_reset_on_fork')])},
7843                         {'call': 'capget',
7844                          'reason': set([('task_struct',
7845                                          'sched_reset_on_fork')])},
7846                         {'call': 'sched_setaffinity',
7847                          'reason': set([('task_struct',
```



```
7848                                          'sched_reset_on_fork')])},
7849                         {'call': 'signal',
7850                          'reason': set([('task_struct',
7851                                          'sched_reset_on_fork')])},
7852                         {'call': 'semtimedop',
7853                          'reason': set([('task_struct',
7854                                          'sched_reset_on_fork')])},
7855                         {'call': 'umount',
7856                          'reason': set([('task_struct',
7857                                          'sched_reset_on_fork')])},
7858                         {'call': 'sched_rr_get_interval',
7859                          'reason': set([('task_struct',
7860                                          'sched_reset_on_fork')])},
7861                         {'call': 'rt_sigprocmask',
7862                          'reason': set([('task_struct',
7863                                          'sched_reset_on_fork')])},
7864                         {'call': 'setsid',
7865                          'reason': set([('task_struct',
7866                                          'sched_reset_on_fork')])},
7867                         {'call': 'sigaltstack',
7868                          'reason': set([('task_struct',
7869                                          'sched_reset_on_fork')])},
7870                         {'call': 'sched_setattr',
7871                          'reason': set([('task_struct',
7872                                          'sched_reset_on_fork')])},
7873                         {'call': 'migrate_pages',
7874                          'reason': set([('task_struct',
7875                                          'sched_reset_on_fork')])},
7876                         {'call': 'getitimer',
7877                          'reason': set([('task_struct',
7878                                          'sched_reset_on_fork')])},
7879                         {'call': 'setpgid',
7880                          'reason': set([('task_struct',
7881                                          'sched_reset_on_fork')])},
7882                         {'call': 'getsid',
7883                          'reason': set([('task_struct',
7884                                          'sched_reset_on_fork')])},
7885                         {'call': 'prlimit64',
7886                          'reason': set([('task_struct',
7887                                          'sched_reset_on_fork')])},
7888                         {'call': 'perf_event_open',
7889                          'reason': set([('task_struct',
7890                                          'sched_reset_on_fork')])},
7891                         {'call': 'rt_sigaction',
7892                          'reason': set([('task_struct',
7893                                          'sched_reset_on_fork')])},
7894                         {'call': 'getpgid',
7895                          'reason': set([('task_struct',
7896                                          'sched_reset_on_fork')])},
7897                         {'call': 'getpriority',
7898                          'reason': set([('task_struct',
7899                                          'sched_reset_on_fork')])},
7900                         {'call': 'sigaction',
7901                          'reason': set([('task_struct',
7902                                          'sched_reset_on_fork')])},
7903                         {'call': 'setns',
7904                          'reason': set([('task_struct',
7905                                          'sched_reset_on_fork')])},
7906                         {'call': 'fork',
7907                          'reason': set([('task_struct',
7908                                          'sched_reset_on_fork')])},
7909                         {'call': 'get_robust_list',
7910                          'reason': set([('task_struct',
7911                                          'sched_reset_on_fork')])},
7912                         {'call': 'mq_timedsend',
7913                          'reason': set([('task_struct',
```

```
7914                                        'sched_reset_on_fork')])},
7915                          {'call': 'ptrace',
7916                           'reason': set([('task_struct',
7917                                           'sched_reset_on_fork')])},
7918                          {'call': 'sched_getattr',
7919                           'reason': set([('task_struct',
7920                                           'sched_reset_on_fork')])},
7921                          {'call': 'getrusage',
7922                           'reason': set([('task_struct',
7923                                           'sched_reset_on_fork')])},
7924                          {'call': 'sched_setscheduler',
7925                           'reason': set([('task_struct',
7926                                           'sched_reset_on_fork')])},
7927                          {'call': 'setitimer',
7928                           'reason': set([('task_struct',
7929                                           'sched_reset_on_fork')])},
7930                          {'call': 'ioprio_get',
7931                           'reason': set([('task_struct',
7932                                           'sched_reset_on_fork')])},
7933                          {'call': 'vfork',
7934                           'reason': set([('task_struct',
7935                                           'sched_reset_on_fork')])},
7936                          {'call': 'prctl',
7937                           'reason': set([('task_struct',
7938                                           'sched_reset_on_fork')])},
7939                          {'call': 'move_pages',
7940                           'reason': set([('task_struct',
7941                                           'sched_reset_on_fork')])},
7942                          {'call': 'setpriority',
7943                           'reason': set([('task_struct',
7944                                           'sched_reset_on_fork')])},
7945                          {'call': 'clone',
7946                           'reason': set([('task_struct',
7947                                           'sched_reset_on_fork')])},
7948                          {'call': 'sched_getparam',
7949                           'reason': set([('task_struct',
7950                                           'sched_reset_on_fork')])}],
7951  'sched_setaffinity': [{'call': 'keyctl',
7952                         'reason': set([('task_struct', 'flags')])},
7953                        {'call': 'rt_sigtimedwait',
7954                         'reason': set([('task_struct', 'flags')])},
7955                        {'call': 'msgrcv',
7956                         'reason': set([('task_struct', 'flags')])},
7957                        {'call': 'kill',
7958                         'reason': set([('task_struct', 'flags')])},
7959                        {'call': 'sched_getaffinity',
7960                         'reason': set([('task_struct', 'flags')])},
7961                        {'call': 'sched_setparam',
7962                         'reason': set([('task_struct', 'flags')])},
7963                        {'call': 'ioprio_set',
7964                         'reason': set([('task_struct', 'flags')])},
7965                        {'call': 'getppid',
7966                         'reason': set([('task_struct', 'flags')])},
7967                        {'call': 'mq_timedreceive',
7968                         'reason': set([('task_struct', 'flags')])},
7969                        {'call': 'capget',
7970                         'reason': set([('task_struct', 'flags')])},
7971                        {'call': 'signal',
7972                         'reason': set([('task_struct', 'flags')])},
7973                        {'call': 'setreuid',
7974                         'reason': set([('task_struct', 'flags')])},
7975                        {'call': 'semtimedop',
7976                         'reason': set([('task_struct', 'flags')])},
7977                        {'call': 'umount',
7978                         'reason': set([('task_struct', 'flags')])},
7979                        {'call': 'sched_rr_get_interval',
```



```
7980                         'reason': set([('task_struct', 'flags')])},
7981                        {'call': 'rt_sigprocmask',
7982                         'reason': set([('task_struct', 'flags')])},
7983                        {'call': 'setsid',
7984                         'reason': set([('task_struct', 'flags')])},
7985                        {'call': 'sigaltstack',
7986                         'reason': set([('task_struct', 'flags')])},
7987                        {'call': 'sched_setattr',
7988                         'reason': set([('task_struct', 'flags')])},
7989                        {'call': 'migrate_pages',
7990                         'reason': set([('task_struct', 'flags')])},
7991                        {'call': 'getitimer',
7992                         'reason': set([('task_struct', 'flags')])},
7993                        {'call': 'setpgid',
7994                         'reason': set([('task_struct', 'flags')])},
7995                        {'call': 'getsid',
7996                         'reason': set([('task_struct', 'flags')])},
7997                        {'call': 'prlimit64',
7998                         'reason': set([('task_struct', 'flags')])},
7999                        {'call': 'perf_event_open',
8000                         'reason': set([('task_struct', 'flags')])},
8001                        {'call': 'rt_sigaction',
8002                         'reason': set([('task_struct', 'flags')])},
8003                        {'call': 'getpgid',
8004                         'reason': set([('task_struct', 'flags')])},
8005                        {'call': 'getpriority',
8006                         'reason': set([('task_struct', 'flags')])},
8007                        {'call': 'sigaction',
8008                         'reason': set([('task_struct', 'flags')])},
8009                        {'call': 'setns',
8010                         'reason': set([('task_struct', 'flags')])},
8011                        {'call': 'fork',
8012                         'reason': set([('task_struct', 'flags')])},
8013                        {'call': 'get_robust_list',
8014                         'reason': set([('task_struct', 'flags')])},
8015                        {'call': 'mq_timedsend',
8016                         'reason': set([('task_struct', 'flags')])},
8017                        {'call': 'sched_getscheduler',
8018                         'reason': set([('task_struct', 'flags')])},
8019                        {'call': 'ptrace',
8020                         'reason': set([('task_struct', 'flags')])},
8021                        {'call': 'sched_getattr',
8022                         'reason': set([('task_struct', 'flags')])},
8023                        {'call': 'getrusage',
8024                         'reason': set([('task_struct', 'flags')])},
8025                        {'call': 'sched_setscheduler',
8026                         'reason': set([('task_struct', 'flags')])},
8027                        {'call': 'setresuid',
8028                         'reason': set([('task_struct', 'flags')])},
8029                        {'call': 'setitimer',
8030                         'reason': set([('task_struct', 'flags')])},
8031                        {'call': 'ioprio_get',
8032                         'reason': set([('task_struct', 'flags')])},
8033                        {'call': 'vfork',
8034                         'reason': set([('task_struct', 'flags')])},
8035                        {'call': 'setuid',
8036                         'reason': set([('task_struct', 'flags')])},
8037                        {'call': 'prctl',
8038                         'reason': set([('task_struct', 'flags')])},
8039                        {'call': 'move_pages',
8040                         'reason': set([('task_struct', 'flags')])},
8041                        {'call': 'setpriority',
8042                         'reason': set([('task_struct', 'flags')])},
8043                        {'call': 'clone',
8044                         'reason': set([('task_struct', 'flags')])},
8045                        {'call': 'sched_getparam',
```

```
                      'reason': set([('task_struct', 'flags')])}],
 'sched_setattr': [{'call': 'keyctl',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'rt_sigtimedwait',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'msgrcv',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_getaffinity',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_setparam',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ioprio_set',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getppid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ioperm',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'mq_timedreceive',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'capget',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_setaffinity',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'signal',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'semtimedop',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'umount',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_rr_get_interval',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'rt_sigprocmask',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setsid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sigaltstack',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'migrate_pages',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getitimer',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setpgid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getsid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'prlimit64',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'perf_event_open',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'rt_sigaction',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getpgid',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'getpriority',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sigaction',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setns',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'get_robust_list',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'mq_timedsend',
                    'reason': set([('mm_segment_t', 'seg')])},
```



```
                   {'call': 'sched_getscheduler',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ptrace',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_getattr',
                    'reason': set([('mm_segment_t', 'seg'),
                                   ('sched_attr', 'sched_policy')])},
                   {'call': 'getrusage',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_setscheduler',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setitimer',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'ioprio_get',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'vfork',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'prctl',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'move_pages',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'setpriority',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'clone',
                    'reason': set([('mm_segment_t', 'seg')])},
                   {'call': 'sched_getparam',
                    'reason': set([('mm_segment_t', 'seg')])}],
 'select': [{'call': 'keyctl',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('task_struct', 'personality'),
                            ('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'msgrcv',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'fadvise64_64',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'mq_unlink',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
            {'call': 'swapoff',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'sched_getaffinity',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'sched_setparam',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'fchmod',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'memfd_create',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'ioprio_set',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'personality',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'readlinkat',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'io_getevents',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'getppid',
```

```
8178             'reason': set([('task_struct', 'personality')])},
8179            {'call': 'fchown',
8180             'reason': set([('timespec', 'tv_nsec'),
8181                            ('timespec', 'tv_sec')])},
8182            {'call': 'mq_timedreceive',
8183             'reason': set([('task_struct', 'personality'),
8184                            ('timespec', 'tv_nsec'),
8185                            ('timespec', 'tv_sec')])},
8186            {'call': 'capget',
8187             'reason': set([('task_struct', 'personality')])},
8188            {'call': 'utime',
8189             'reason': set([('timespec', 'tv_nsec'),
8190                            ('timespec', 'tv_sec')])},
8191            {'call': 'sched_setaffinity',
8192             'reason': set([('task_struct', 'personality')])},
8193            {'call': 'signal',
8194             'reason': set([('task_struct', 'personality')])},
8195            {'call': 'semtimedop',
8196             'reason': set([('task_struct', 'personality'),
8197                            ('timespec', 'tv_nsec'),
8198                            ('timespec', 'tv_sec')])},
8199            {'call': 'umount',
8200             'reason': set([('task_struct', 'personality')])},
8201            {'call': 'settimeofday',
8202             'reason': set([('timespec', 'tv_nsec'),
8203                            ('timespec', 'tv_sec')])},
8204            {'call': 'sched_rr_get_interval',
8205             'reason': set([('task_struct', 'personality'),
8206                            ('timespec', 'tv_nsec'),
8207                            ('timespec', 'tv_sec')])},
8208            {'call': 'timerfd_gettime',
8209             'reason': set([('timespec', 'tv_nsec'),
8210                            ('timespec', 'tv_sec')])},
8211            {'call': 'pselect6',
8212             'reason': set([('timespec', 'tv_nsec'),
8213                            ('timespec', 'tv_sec')])},
8214            {'call': 'uselib',
8215             'reason': set([('timespec', 'tv_nsec'),
8216                            ('timespec', 'tv_sec')])},
8217            {'call': 'rt_sigprocmask',
8218             'reason': set([('task_struct', 'personality')])},
8219            {'call': 'setsid',
8220             'reason': set([('task_struct', 'personality')])},
8221            {'call': 'sigaltstack',
8222             'reason': set([('task_struct', 'personality')])},
8223            {'call': 'sched_setattr',
8224             'reason': set([('task_struct', 'personality')])},
8225            {'call': 'migrate_pages',
8226             'reason': set([('task_struct', 'personality')])},
8227            {'call': 'getitimer',
8228             'reason': set([('task_struct', 'personality')])},
8229            {'call': 'fchmodat',
8230             'reason': set([('timespec', 'tv_nsec'),
8231                            ('timespec', 'tv_sec')])},
8232            {'call': 'setpgid',
8233             'reason': set([('task_struct', 'personality')])},
8234            {'call': 'inotify_add_watch',
8235             'reason': set([('timespec', 'tv_nsec'),
8236                            ('timespec', 'tv_sec')])},
8237            {'call': 'timer_settime',
8238             'reason': set([('timespec', 'tv_nsec'),
8239                            ('timespec', 'tv_sec')])},
8240            {'call': 'ftruncate',
8241             'reason': set([('timespec', 'tv_nsec'),
8242                            ('timespec', 'tv_sec')])},
8243            {'call': 'timer_gettime',
```



```
8244             'reason': set([('timespec', 'tv_nsec'),
8245                            ('timespec', 'tv_sec')])},
8246            {'call': 'getsid',
8247             'reason': set([('task_struct', 'personality')])},
8248            {'call': 'ioctl',
8249             'reason': set([('timespec', 'tv_nsec'),
8250                            ('timespec', 'tv_sec')])},
8251            {'call': 'prlimit64',
8252             'reason': set([('task_struct', 'personality')])},
8253            {'call': 'perf_event_open',
8254             'reason': set([('task_struct', 'personality')])},
8255            {'call': 'linkat',
8256             'reason': set([('timespec', 'tv_nsec'),
8257                            ('timespec', 'tv_sec')])},
8258            {'call': 'stime',
8259             'reason': set([('timespec', 'tv_nsec'),
8260                            ('timespec', 'tv_sec')])},
8261            {'call': 'rt_sigaction',
8262             'reason': set([('task_struct', 'personality')])},
8263            {'call': 'futimesat',
8264             'reason': set([('timespec', 'tv_nsec'),
8265                            ('timespec', 'tv_sec')])},
8266            {'call': 'getpgid',
8267             'reason': set([('task_struct', 'personality')])},
8268            {'call': 'poll',
8269             'reason': set([('timespec', 'tv_nsec'),
8270                            ('timespec', 'tv_sec')])},
8271            {'call': 'unlink',
8272             'reason': set([('timespec', 'tv_nsec'),
8273                            ('timespec', 'tv_sec')])},
8274            {'call': 'getpriority',
8275             'reason': set([('task_struct', 'personality')])},
8276            {'call': 'sigaction',
8277             'reason': set([('task_struct', 'personality')])},
8278            {'call': 'nanosleep',
8279             'reason': set([('timespec', 'tv_nsec'),
8280                            ('timespec', 'tv_sec')])},
8281            {'call': 'mq_getsetattr',
8282             'reason': set([('timespec', 'tv_nsec'),
8283                            ('timespec', 'tv_sec')])},
8284            {'call': 'faccessat',
8285             'reason': set([('timespec', 'tv_nsec'),
8286                            ('timespec', 'tv_sec')])},
8287            {'call': 'setns',
8288             'reason': set([('task_struct', 'personality')])},
8289            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
8290            {'call': 'get_robust_list',
8291             'reason': set([('task_struct', 'personality')])},
8292            {'call': 'mq_timedsend',
8293             'reason': set([('task_struct', 'personality'),
8294                            ('timespec', 'tv_nsec'),
8295                            ('timespec', 'tv_sec')])},
8296            {'call': 'sched_getscheduler',
8297             'reason': set([('task_struct', 'personality')])},
8298            {'call': 'ptrace',
8299             'reason': set([('task_struct', 'personality')])},
8300            {'call': 'swapon',
8301             'reason': set([('timespec', 'tv_nsec'),
8302                            ('timespec', 'tv_sec')])},
8303            {'call': 'epoll_wait',
8304             'reason': set([('timespec', 'tv_nsec'),
8305                            ('timespec', 'tv_sec')])},
8306            {'call': 'sched_getattr',
8307             'reason': set([('task_struct', 'personality')])},
8308            {'call': 'fchownat',
8309             'reason': set([('timespec', 'tv_nsec'),
```

```
                            ('timespec', 'tv_sec')])},
            {'call': 'getrusage',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'timerfd_settime',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'sched_setscheduler',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'setitimer',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'ioprio_get',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'vfork',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'prctl',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'move_pages',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'setpriority',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'mq_notify',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'sendfile',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'clone',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'clock_nanosleep',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'unlinkat',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'sched_getparam',
             'reason': set([('task_struct', 'personality')])},
            {'call': 'futex',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'recvmmsg',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'sendfile64',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])},
            {'call': 'ppoll',
             'reason': set([('timespec', 'tv_nsec'),
                            ('timespec', 'tv_sec')])}],
 'semctl': [{'call': 'semtimedop',
             'reason': set([('sem_array', 'sem_nsems')])}],
 'semtimedop': [{'call': 'rt_sigtimedwait',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'msgrcv',
                 'reason': set([('kern_ipc_perm', 'deleted')])},
                {'call': 'fadvise64_64',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'mq_unlink',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'swapoff',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'fchmod',
                 'reason': set([('timespec', 'tv_nsec'),
```



```
                                ('timespec', 'tv_sec')])},
                {'call': 'memfd_create',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'readlinkat',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'io_getevents',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'fchown',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'mq_timedreceive',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'utime',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'settimeofday',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'sched_rr_get_interval',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'timerfd_gettime',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'semctl',
                 'reason': set([('kern_ipc_perm', 'deleted'),
                                ('sem_array', 'complex_count'),
                                ('sem_array', 'sem_nsems'),
                                ('sem_array', 'use_global_lock'),
                                ('sem_undo', 'semid')])},
                {'call': 'pselect6',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'uselib',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'msgctl',
                 'reason': set([('kern_ipc_perm', 'deleted')])},
                {'call': 'fchmodat',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'inotify_add_watch',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'timer_settime',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'ftruncate',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'timer_gettime',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'shmat',
                 'reason': set([('kern_ipc_perm', 'deleted')])},
                {'call': 'ioctl',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'linkat',
                 'reason': set([('timespec', 'tv_nsec'),
                                ('timespec', 'tv_sec')])},
                {'call': 'stime',
```

```
8442                  'reason': set([('timespec', 'tv_nsec'),
8443                                 ('timespec', 'tv_sec')])},
8444                {'call': 'futimesat',
8445                 'reason': set([('timespec', 'tv_nsec'),
8446                                ('timespec', 'tv_sec')])},
8447                {'call': 'poll',
8448                 'reason': set([('timespec', 'tv_nsec'),
8449                                ('timespec', 'tv_sec')])},
8450                {'call': 'select',
8451                 'reason': set([('timespec', 'tv_nsec'),
8452                                ('timespec', 'tv_sec')])},
8453                {'call': 'unlink',
8454                 'reason': set([('timespec', 'tv_nsec'),
8455                                ('timespec', 'tv_sec')])},
8456                {'call': 'nanosleep',
8457                 'reason': set([('timespec', 'tv_nsec'),
8458                                ('timespec', 'tv_sec')])},
8459                {'call': 'mq_getsetattr',
8460                 'reason': set([('timespec', 'tv_nsec'),
8461                                ('timespec', 'tv_sec')])},
8462                {'call': 'faccessat',
8463                 'reason': set([('timespec', 'tv_nsec'),
8464                                ('timespec', 'tv_sec')])},
8465                {'call': 'mq_timedsend',
8466                 'reason': set([('timespec', 'tv_nsec'),
8467                                ('timespec', 'tv_sec')])},
8468                {'call': 'shmctl',
8469                 'reason': set([('kern_ipc_perm', 'deleted')])},
8470                {'call': 'swapon',
8471                 'reason': set([('timespec', 'tv_nsec'),
8472                                ('timespec', 'tv_sec')])},
8473                {'call': 'epoll_wait',
8474                 'reason': set([('timespec', 'tv_nsec'),
8475                                ('timespec', 'tv_sec')])},
8476                {'call': 'fchownat',
8477                 'reason': set([('timespec', 'tv_nsec'),
8478                                ('timespec', 'tv_sec')])},
8479                {'call': 'timerfd_settime',
8480                 'reason': set([('timespec', 'tv_nsec'),
8481                                ('timespec', 'tv_sec')])},
8482                {'call': 'msgsnd',
8483                 'reason': set([('kern_ipc_perm', 'deleted')])},
8484                {'call': 'mq_notify',
8485                 'reason': set([('timespec', 'tv_nsec'),
8486                                ('timespec', 'tv_sec')])},
8487                {'call': 'sendfile',
8488                 'reason': set([('timespec', 'tv_nsec'),
8489                                ('timespec', 'tv_sec')])},
8490                {'call': 'clock_nanosleep',
8491                 'reason': set([('timespec', 'tv_nsec'),
8492                                ('timespec', 'tv_sec')])},
8493                {'call': 'unlinkat',
8494                 'reason': set([('timespec', 'tv_nsec'),
8495                                ('timespec', 'tv_sec')])},
8496                {'call': 'futex',
8497                 'reason': set([('timespec', 'tv_nsec'),
8498                                ('timespec', 'tv_sec')])},
8499                {'call': 'recvmmsg',
8500                 'reason': set([('timespec', 'tv_nsec'),
8501                                ('timespec', 'tv_sec')])},
8502                {'call': 'sendfile64',
8503                 'reason': set([('timespec', 'tv_nsec'),
8504                                ('timespec', 'tv_sec')])},
8505                {'call': 'ppoll',
8506                 'reason': set([('timespec', 'tv_nsec'),
8507                                ('timespec', 'tv_sec')])}],
```



```
8508  'sendfile': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
8509               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
8510               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
8511               {'call': 'remap_file_pages',
8512                'reason': set([('file', 'f_mode')])},
8513               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
8514               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
8515               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
8516               {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
8517               {'call': 'flock', 'reason': set([('file', 'f_mode')])},
8518               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
8519               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
8520               {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
8521               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
8522               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
8523               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
8524               {'call': 'perf_event_open',
8525                'reason': set([('file', 'f_mode')])},
8526               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
8527               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
8528               {'call': 'open', 'reason': set([('file', 'f_mode')])},
8529               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
8530               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
8531               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
8532               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
8533               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
8534               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
8535               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
8536               {'call': 'msync', 'reason': set([('file', 'f_mode')])},
8537               {'call': 'open_by_handle_at',
8538                'reason': set([('file', 'f_mode')])}],
8539  'sendfile64': [{'call': 'fadvise64_64', 'reason': set([('file', 'f_mode')])},
8540                 {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
8541                 {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
8542                 {'call': 'remap_file_pages',
8543                  'reason': set([('file', 'f_mode')])},
8544                 {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
8545                 {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
8546                 {'call': 'epoll_create1',
8547                  'reason': set([('file', 'f_mode')])},
8548                 {'call': 'epoll_ctl', 'reason': set([('file', 'f_mode')])},
8549                 {'call': 'flock', 'reason': set([('file', 'f_mode')])},
8550                 {'call': 'openat', 'reason': set([('file', 'f_mode')])},
8551                 {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
8552                 {'call': 'accept4', 'reason': set([('file', 'f_mode')])},
8553                 {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
8554                 {'call': 'socket', 'reason': set([('file', 'f_mode')])},
8555                 {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
8556                 {'call': 'perf_event_open',
8557                  'reason': set([('file', 'f_mode')])},
8558                 {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
8559                 {'call': 'acct', 'reason': set([('file', 'f_mode')])},
8560                 {'call': 'open', 'reason': set([('file', 'f_mode')])},
8561                 {'call': 'dup', 'reason': set([('file', 'f_mode')])},
8562                 {'call': 'setns', 'reason': set([('file', 'f_mode')])},
8563                 {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
8564                 {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
8565                 {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
8566                 {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
8567                 {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
8568                 {'call': 'msync', 'reason': set([('file', 'f_mode')])},
8569                 {'call': 'open_by_handle_at',
8570                  'reason': set([('file', 'f_mode')])}],
8571  'sendto': [{'call': 'swapoff', 'reason': set([('file', 'f_flags')])},
8572             {'call': 'memfd_create', 'reason': set([('file', 'f_flags')])},
8573             {'call': 'remap_file_pages',
```

```
8574              'reason': set([('file', 'f_flags')])},
8575             {'call': 'dup3', 'reason': set([('file', 'f_flags')])},
8576             {'call': 'socketpair', 'reason': set([('file', 'f_flags')])},
8577             {'call': 'epoll_create1', 'reason': set([('file', 'f_flags')])},
8578             {'call': 'epoll_ctl', 'reason': set([('file', 'f_flags')])},
8579             {'call': 'flock', 'reason': set([('file', 'f_flags')])},
8580             {'call': 'openat', 'reason': set([('file', 'f_flags')])},
8581             {'call': 'uselib', 'reason': set([('file', 'f_flags')])},
8582             {'call': 'accept4', 'reason': set([('file', 'f_flags')])},
8583             {'call': 'shmat', 'reason': set([('file', 'f_flags')])},
8584             {'call': 'socket', 'reason': set([('file', 'f_flags')])},
8585             {'call': 'pipe2', 'reason': set([('file', 'f_flags')])},
8586             {'call': 'perf_event_open', 'reason': set([('file', 'f_flags')])},
8587             {'call': 'shmdt', 'reason': set([('file', 'f_flags')])},
8588             {'call': 'acct', 'reason': set([('file', 'f_flags')])},
8589             {'call': 'open', 'reason': set([('file', 'f_flags')])},
8590             {'call': 'mq_getsetattr', 'reason': set([('file', 'f_flags')])},
8591             {'call': 'dup', 'reason': set([('file', 'f_flags')])},
8592             {'call': 'setns', 'reason': set([('file', 'f_flags')])},
8593             {'call': 'shmctl', 'reason': set([('file', 'f_flags')])},
8594             {'call': 'swapon', 'reason': set([('file', 'f_flags')])},
8595             {'call': 'eventfd2', 'reason': set([('file', 'f_flags')])},
8596             {'call': 'mmap_pgoff', 'reason': set([('file', 'f_flags')])},
8597             {'call': 'mq_open', 'reason': set([('file', 'f_flags')])},
8598             {'call': 'msync', 'reason': set([('file', 'f_flags')])},
8599             {'call': 'open_by_handle_at',
8600              'reason': set([('file', 'f_flags')])}],
8601  'set_mempolicy': [{'call': 'get_mempolicy',
8602                     'reason': set([('mempolicy', 'mode')])},
8603                    {'call': 'mbind', 'reason': set([('mempolicy', 'mode')])}],
8604  'set_thread_area': [{'call': 'keyctl',
8605                       'reason': set([('thread_struct', 'fsindex'),
8606                                      ('thread_struct', 'gsindex')])},
8607                      {'call': 'rt_sigtimedwait',
8608                       'reason': set([('thread_struct', 'fsindex'),
8609                                      ('thread_struct', 'gsindex')])},
8610                      {'call': 'iopl',
8611                       'reason': set([('thread_struct', 'fsindex'),
8612                                      ('thread_struct', 'gsindex')])},
8613                      {'call': 'msgrcv',
8614                       'reason': set([('thread_struct', 'fsindex'),
8615                                      ('thread_struct', 'gsindex')])},
8616                      {'call': 'kill',
8617                       'reason': set([('thread_struct', 'fsindex'),
8618                                      ('thread_struct', 'gsindex')])},
8619                      {'call': 'sched_getaffinity',
8620                       'reason': set([('thread_struct', 'fsindex'),
8621                                      ('thread_struct', 'gsindex')])},
8622                      {'call': 'arch_prctl',
8623                       'reason': set([('thread_struct', 'fsindex'),
8624                                      ('thread_struct', 'gsindex')])},
8625                      {'call': 'sched_setparam',
8626                       'reason': set([('thread_struct', 'fsindex'),
8627                                      ('thread_struct', 'gsindex')])},
8628                      {'call': 'ioprio_set',
8629                       'reason': set([('thread_struct', 'fsindex'),
8630                                      ('thread_struct', 'gsindex')])},
8631                      {'call': 'getppid',
8632                       'reason': set([('thread_struct', 'fsindex'),
8633                                      ('thread_struct', 'gsindex')])},
8634                      {'call': 'ioperm',
8635                       'reason': set([('thread_struct', 'fsindex'),
8636                                      ('thread_struct', 'gsindex')])},
8637                      {'call': 'mq_timedreceive',
8638                       'reason': set([('thread_struct', 'fsindex'),
8639                                      ('thread_struct', 'gsindex')])},
```



```
8640                      {'call': 'capget',
8641                       'reason': set([('thread_struct', 'fsindex'),
8642                                      ('thread_struct', 'gsindex')])},
8643                      {'call': 'sched_setaffinity',
8644                       'reason': set([('thread_struct', 'fsindex'),
8645                                      ('thread_struct', 'gsindex')])},
8646                      {'call': 'signal',
8647                       'reason': set([('thread_struct', 'fsindex'),
8648                                      ('thread_struct', 'gsindex')])},
8649                      {'call': 'semtimedop',
8650                       'reason': set([('thread_struct', 'fsindex'),
8651                                      ('thread_struct', 'gsindex')])},
8652                      {'call': 'umount',
8653                       'reason': set([('thread_struct', 'fsindex'),
8654                                      ('thread_struct', 'gsindex')])},
8655                      {'call': 'sched_rr_get_interval',
8656                       'reason': set([('thread_struct', 'fsindex'),
8657                                      ('thread_struct', 'gsindex')])},
8658                      {'call': 'rt_sigprocmask',
8659                       'reason': set([('thread_struct', 'fsindex'),
8660                                      ('thread_struct', 'gsindex')])},
8661                      {'call': 'setsid',
8662                       'reason': set([('thread_struct', 'fsindex'),
8663                                      ('thread_struct', 'gsindex')])},
8664                      {'call': 'sigaltstack',
8665                       'reason': set([('thread_struct', 'fsindex'),
8666                                      ('thread_struct', 'gsindex')])},
8667                      {'call': 'sched_setattr',
8668                       'reason': set([('thread_struct', 'fsindex'),
8669                                      ('thread_struct', 'gsindex')])},
8670                      {'call': 'migrate_pages',
8671                       'reason': set([('thread_struct', 'fsindex'),
8672                                      ('thread_struct', 'gsindex')])},
8673                      {'call': 'getitimer',
8674                       'reason': set([('thread_struct', 'fsindex'),
8675                                      ('thread_struct', 'gsindex')])},
8676                      {'call': 'setpgid',
8677                       'reason': set([('thread_struct', 'fsindex'),
8678                                      ('thread_struct', 'gsindex')])},
8679                      {'call': 'getsid',
8680                       'reason': set([('thread_struct', 'fsindex'),
8681                                      ('thread_struct', 'gsindex')])},
8682                      {'call': 'prlimit64',
8683                       'reason': set([('thread_struct', 'fsindex'),
8684                                      ('thread_struct', 'gsindex')])},
8685                      {'call': 'perf_event_open',
8686                       'reason': set([('thread_struct', 'fsindex'),
8687                                      ('thread_struct', 'gsindex')])},
8688                      {'call': 'rt_sigaction',
8689                       'reason': set([('thread_struct', 'fsindex'),
8690                                      ('thread_struct', 'gsindex')])},
8691                      {'call': 'getpgid',
8692                       'reason': set([('thread_struct', 'fsindex'),
8693                                      ('thread_struct', 'gsindex')])},
8694                      {'call': 'getpriority',
8695                       'reason': set([('thread_struct', 'fsindex'),
8696                                      ('thread_struct', 'gsindex')])},
8697                      {'call': 'sigaction',
8698                       'reason': set([('thread_struct', 'fsindex'),
8699                                      ('thread_struct', 'gsindex')])},
8700                      {'call': 'setns',
8701                       'reason': set([('thread_struct', 'fsindex'),
8702                                      ('thread_struct', 'gsindex')])},
8703                      {'call': 'fork',
8704                       'reason': set([('thread_struct', 'fsindex'),
8705                                      ('thread_struct', 'gsindex')])},
```

```
8706                     {'call': 'get_robust_list',
8707                      'reason': set([('thread_struct', 'fsindex'),
8708                                     ('thread_struct', 'gsindex')])},
8709                     {'call': 'mq_timedsend',
8710                      'reason': set([('thread_struct', 'fsindex'),
8711                                     ('thread_struct', 'gsindex')])},
8712                     {'call': 'sched_getscheduler',
8713                      'reason': set([('thread_struct', 'fsindex'),
8714                                     ('thread_struct', 'gsindex')])},
8715                     {'call': 'ptrace',
8716                      'reason': set([('thread_struct', 'fsindex'),
8717                                     ('thread_struct', 'gsindex')])},
8718                     {'call': 'sched_getattr',
8719                      'reason': set([('thread_struct', 'fsindex'),
8720                                     ('thread_struct', 'gsindex')])},
8721                     {'call': 'getrusage',
8722                      'reason': set([('thread_struct', 'fsindex'),
8723                                     ('thread_struct', 'gsindex')])},
8724                     {'call': 'sched_setscheduler',
8725                      'reason': set([('thread_struct', 'fsindex'),
8726                                     ('thread_struct', 'gsindex')])},
8727                     {'call': 'setitimer',
8728                      'reason': set([('thread_struct', 'fsindex'),
8729                                     ('thread_struct', 'gsindex')])},
8730                     {'call': 'ioprio_get',
8731                      'reason': set([('thread_struct', 'fsindex'),
8732                                     ('thread_struct', 'gsindex')])},
8733                     {'call': 'vfork',
8734                      'reason': set([('thread_struct', 'fsindex'),
8735                                     ('thread_struct', 'gsindex')])},
8736                     {'call': 'prctl',
8737                      'reason': set([('thread_struct', 'fsindex'),
8738                                     ('thread_struct', 'gsindex')])},
8739                     {'call': 'move_pages',
8740                      'reason': set([('thread_struct', 'fsindex'),
8741                                     ('thread_struct', 'gsindex')])},
8742                     {'call': 'setpriority',
8743                      'reason': set([('thread_struct', 'fsindex'),
8744                                     ('thread_struct', 'gsindex')])},
8745                     {'call': 'clone',
8746                      'reason': set([('thread_struct', 'fsindex'),
8747                                     ('thread_struct', 'gsindex')])},
8748                     {'call': 'sched_getparam',
8749                      'reason': set([('thread_struct', 'fsindex'),
8750                                     ('thread_struct', 'gsindex')])}],
8751  'set_trip_temp': [{'call': 'get_trip_temp',
8752                     'reason': set([('pkg_device', 'cpu'),
8753                                    ('pkg_device', 'tj_max')])},
8754                    {'call': 'get_curr_temp',
8755                     'reason': set([('pkg_device', 'cpu'),
8756                                    ('pkg_device', 'tj_max')])}],
8757  'setgroups16': [{'call': 'setgroups',
8758                   'reason': set([('group_info', 'ngroups')])}],
8759  'setitimer': [{'call': 'settimeofday',
8760                 'reason': set([('timeval', 'tv_sec'),
8761                                ('timeval', 'tv_usec')])},
8762                {'call': 'timer_create',
8763                 'reason': set([('signal_struct', 'it_real_incr')])},
8764                {'call': 'adjtimex',
8765                 'reason': set([('timeval', 'tv_sec'),
8766                                ('timeval', 'tv_usec')])},
8767                {'call': 'waitid',
8768                 'reason': set([('timeval', 'tv_sec'),
8769                                ('timeval', 'tv_usec')])},
8770                {'call': 'getitimer',
8771                 'reason': set([('timeval', 'tv_sec'),
```



```
8772                                ('timeval', 'tv_usec')])},
8773                {'call': 'select',
8774                 'reason': set([('timeval', 'tv_sec'),
8775                                ('timeval', 'tv_usec')])},
8776                {'call': 'exit_group',
8777                 'reason': set([('signal_struct', 'it_real_incr')])},
8778                {'call': 'wait4',
8779                 'reason': set([('timeval', 'tv_sec'),
8780                                ('timeval', 'tv_usec')])},
8781                {'call': 'getrusage',
8782                 'reason': set([('timeval', 'tv_sec'),
8783                                ('timeval', 'tv_usec')])},
8784                {'call': 'clock_adjtime',
8785                 'reason': set([('timeval', 'tv_sec'),
8786                                ('timeval', 'tv_usec')])},
8787                {'call': 'alarm',
8788                 'reason': set([('timeval', 'tv_sec'),
8789                                ('timeval', 'tv_usec')])},
8790                {'call': 'ppoll',
8791                 'reason': set([('timeval', 'tv_sec'),
8792                                ('timeval', 'tv_usec')])}],
8793  'setpgid': [{'call': 'keyctl',
8794               'reason': set([('task_struct', 'exit_signal'),
8795                              ('task_struct', 'flags')])},
8796              {'call': 'rt_sigtimedwait',
8797               'reason': set([('task_struct', 'exit_signal'),
8798                              ('task_struct', 'flags')])},
8799              {'call': 'msgrcv',
8800               'reason': set([('task_struct', 'exit_signal'),
8801                              ('task_struct', 'flags')])},
8802              {'call': 'kill',
8803               'reason': set([('task_struct', 'exit_signal'),
8804                              ('task_struct', 'flags')])},
8805              {'call': 'sched_getaffinity',
8806               'reason': set([('task_struct', 'exit_signal'),
8807                              ('task_struct', 'flags')])},
8808              {'call': 'sched_setparam',
8809               'reason': set([('task_struct', 'exit_signal'),
8810                              ('task_struct', 'flags')])},
8811              {'call': 'ioprio_set',
8812               'reason': set([('task_struct', 'exit_signal'),
8813                              ('task_struct', 'flags')])},
8814              {'call': 'getppid',
8815               'reason': set([('task_struct', 'exit_signal'),
8816                              ('task_struct', 'flags')])},
8817              {'call': 'mq_timedreceive',
8818               'reason': set([('task_struct', 'exit_signal'),
8819                              ('task_struct', 'flags')])},
8820              {'call': 'capget',
8821               'reason': set([('task_struct', 'exit_signal'),
8822                              ('task_struct', 'flags')])},
8823              {'call': 'sched_setaffinity',
8824               'reason': set([('task_struct', 'exit_signal'),
8825                              ('task_struct', 'flags')])},
8826              {'call': 'signal',
8827               'reason': set([('task_struct', 'exit_signal'),
8828                              ('task_struct', 'flags')])},
8829              {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
8830              {'call': 'semtimedop',
8831               'reason': set([('task_struct', 'exit_signal'),
8832                              ('task_struct', 'flags')])},
8833              {'call': 'umount',
8834               'reason': set([('task_struct', 'exit_signal'),
8835                              ('task_struct', 'flags')])},
8836              {'call': 'timer_create',
8837               'reason': set([('signal_struct', 'leader')])},
```

```
8838             {'call': 'sched_rr_get_interval',
8839              'reason': set([('task_struct', 'exit_signal'),
8840                             ('task_struct', 'flags')])},
8841             {'call': 'rt_sigprocmask',
8842              'reason': set([('task_struct', 'exit_signal'),
8843                             ('task_struct', 'flags')])},
8844             {'call': 'setsid',
8845              'reason': set([('signal_struct', 'leader'),
8846                             ('task_struct', 'exit_signal'),
8847                             ('task_struct', 'flags')])},
8848             {'call': 'sigaltstack',
8849              'reason': set([('task_struct', 'exit_signal'),
8850                             ('task_struct', 'flags')])},
8851             {'call': 'sched_setattr',
8852              'reason': set([('task_struct', 'exit_signal'),
8853                             ('task_struct', 'flags')])},
8854             {'call': 'migrate_pages',
8855              'reason': set([('task_struct', 'exit_signal'),
8856                             ('task_struct', 'flags')])},
8857             {'call': 'getitimer',
8858              'reason': set([('task_struct', 'exit_signal'),
8859                             ('task_struct', 'flags')])},
8860             {'call': 'getsid',
8861              'reason': set([('task_struct', 'exit_signal'),
8862                             ('task_struct', 'flags')])},
8863             {'call': 'prlimit64',
8864              'reason': set([('task_struct', 'exit_signal'),
8865                             ('task_struct', 'flags')])},
8866             {'call': 'perf_event_open',
8867              'reason': set([('task_struct', 'exit_signal'),
8868                             ('task_struct', 'flags')])},
8869             {'call': 'rt_sigaction',
8870              'reason': set([('task_struct', 'exit_signal'),
8871                             ('task_struct', 'flags')])},
8872             {'call': 'getpgid',
8873              'reason': set([('task_struct', 'exit_signal'),
8874                             ('task_struct', 'flags')])},
8875             {'call': 'exit_group',
8876              'reason': set([('signal_struct', 'leader')])},
8877             {'call': 'getpriority',
8878              'reason': set([('task_struct', 'exit_signal'),
8879                             ('task_struct', 'flags')])},
8880             {'call': 'sigaction',
8881              'reason': set([('task_struct', 'exit_signal'),
8882                             ('task_struct', 'flags')])},
8883             {'call': 'setns',
8884              'reason': set([('task_struct', 'exit_signal'),
8885                             ('task_struct', 'flags')])},
8886             {'call': 'fork',
8887              'reason': set([('task_struct', 'exit_signal'),
8888                             ('task_struct', 'flags')])},
8889             {'call': 'get_robust_list',
8890              'reason': set([('task_struct', 'exit_signal'),
8891                             ('task_struct', 'flags')])},
8892             {'call': 'mq_timedsend',
8893              'reason': set([('task_struct', 'exit_signal'),
8894                             ('task_struct', 'flags')])},
8895             {'call': 'sched_getscheduler',
8896              'reason': set([('task_struct', 'exit_signal'),
8897                             ('task_struct', 'flags')])},
8898             {'call': 'ptrace',
8899              'reason': set([('task_struct', 'exit_signal'),
8900                             ('task_struct', 'flags')])},
8901             {'call': 'sched_getattr',
8902              'reason': set([('task_struct', 'exit_signal'),
8903                             ('task_struct', 'flags')])},
```



```
8904             {'call': 'getrusage',
8905              'reason': set([('task_struct', 'exit_signal'),
8906                             ('task_struct', 'flags')])},
8907             {'call': 'sched_setscheduler',
8908              'reason': set([('task_struct', 'exit_signal'),
8909                             ('task_struct', 'flags')])},
8910             {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
8911             {'call': 'setitimer',
8912              'reason': set([('task_struct', 'exit_signal'),
8913                             ('task_struct', 'flags')])},
8914             {'call': 'ioprio_get',
8915              'reason': set([('task_struct', 'exit_signal'),
8916                             ('task_struct', 'flags')])},
8917             {'call': 'vfork',
8918              'reason': set([('task_struct', 'exit_signal'),
8919                             ('task_struct', 'flags')])},
8920             {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
8921             {'call': 'prctl',
8922              'reason': set([('task_struct', 'exit_signal'),
8923                             ('task_struct', 'flags')])},
8924             {'call': 'move_pages',
8925              'reason': set([('task_struct', 'exit_signal'),
8926                             ('task_struct', 'flags')])},
8927             {'call': 'setpriority',
8928              'reason': set([('task_struct', 'exit_signal'),
8929                             ('task_struct', 'flags')])},
8930             {'call': 'clone',
8931              'reason': set([('task_struct', 'exit_signal'),
8932                             ('task_struct', 'flags')])},
8933             {'call': 'sched_getparam',
8934              'reason': set([('task_struct', 'exit_signal'),
8935                             ('task_struct', 'flags')])}],
8936 'setrlimit': [{'call': 'old_getrlimit',
8937                'reason': set([('rlimit', 'rlim_cur'),
8938                               ('rlimit', 'rlim_max')])},
8939               {'call': 'prlimit64',
8940                'reason': set([('rlimit', 'rlim_cur'),
8941                               ('rlimit', 'rlim_max')])},
8942               {'call': 'getrlimit',
8943                'reason': set([('compat_rlimit', 'rlim_cur'),
8944                               ('compat_rlimit', 'rlim_max')])}],
8945 'setsid': [{'call': 'timer_create',
8946             'reason': set([('signal_struct', 'leader')])},
8947            {'call': 'exit_group',
8948             'reason': set([('signal_struct', 'leader')])}],
8949 'setsockopt': [{'call': 'accept4',
8950                 'reason': set([('proto_ops', 'compat_setsockopt')])}],
8951 'settimeofday': [{'call': 'adjtimex',
8952                   'reason': set([('timeval', 'tv_sec'),
8953                                  ('timeval', 'tv_usec')])},
8954                  {'call': 'waitid',
8955                   'reason': set([('timeval', 'tv_sec'),
8956                                  ('timeval', 'tv_usec')])},
8957                  {'call': 'getitimer',
8958                   'reason': set([('timeval', 'tv_sec'),
8959                                  ('timeval', 'tv_usec')])},
8960                  {'call': 'select',
8961                   'reason': set([('timeval', 'tv_sec'),
8962                                  ('timeval', 'tv_usec')])},
8963                  {'call': 'wait4',
8964                   'reason': set([('timeval', 'tv_sec'),
8965                                  ('timeval', 'tv_usec')])},
8966                  {'call': 'getrusage',
8967                   'reason': set([('timeval', 'tv_sec'),
8968                                  ('timeval', 'tv_usec')])},
8969                  {'call': 'setitimer',
```

```
                  'reason': set([('timeval', 'tv_sec'),
                                 ('timeval', 'tv_usec')])},
                {'call': 'clock_adjtime',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'alarm',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])},
                {'call': 'ppoll',
                 'reason': set([('timeval', 'tv_sec'),
                                ('timeval', 'tv_usec')])}],
 'shmctl': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'rt_sigtimedwait',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'msgrcv',
             'reason': set([('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode'),
                            ('mm_segment_t', 'seg')])},
            {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getaffinity',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setparam',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'mq_timedreceive',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setaffinity',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'semtimedop',
             'reason': set([('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode'),
                            ('mm_segment_t', 'seg')])},
            {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'semctl',
             'reason': set([('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'msgctl',
             'reason': set([('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setattr',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'migrate_pages',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'shmat',
             'reason': set([('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'perf_event_open',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'rt_sigaction',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
```



```
            {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'get_robust_list',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'mq_timedsend',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getscheduler',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getattr',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setscheduler',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'msgsnd',
             'reason': set([('kern_ipc_perm', 'deleted'),
                            ('kern_ipc_perm', 'mode')])},
            {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getparam',
             'reason': set([('mm_segment_t', 'seg')])}],
 'shmdt': [{'call': 'remap_file_pages',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'brk', 'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'get_mempolicy',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'munlockall',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'pkey_mprotect',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'madvise',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'mprotect',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'mremap',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'prctl', 'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'munlock',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'mincore',
            'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'msync', 'reason': set([('vm_area_struct', 'vm_pgoff')])},
           {'call': 'mlockall',
            'reason': set([('vm_area_struct', 'vm_pgoff')])}],
 'sigaction': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'rt_sigtimedwait',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_getaffinity',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'sched_setparam',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioprio_set',
                'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
               {'call': 'mq_timedreceive',
```

```
9102                 'reason': set([('mm_segment_t', 'seg')])},
9103                {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
9104                {'call': 'sched_setaffinity',
9105                 'reason': set([('mm_segment_t', 'seg')])},
9106                {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
9107                {'call': 'semtimedop',
9108                 'reason': set([('mm_segment_t', 'seg')])},
9109                {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
9110                {'call': 'sched_rr_get_interval',
9111                 'reason': set([('mm_segment_t', 'seg')])},
9112                {'call': 'rt_sigprocmask',
9113                 'reason': set([('mm_segment_t', 'seg')])},
9114                {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
9115                {'call': 'sigaltstack',
9116                 'reason': set([('mm_segment_t', 'seg')])},
9117                {'call': 'sched_setattr',
9118                 'reason': set([('mm_segment_t', 'seg')])},
9119                {'call': 'migrate_pages',
9120                 'reason': set([('mm_segment_t', 'seg')])},
9121                {'call': 'getitimer',
9122                 'reason': set([('mm_segment_t', 'seg')])},
9123                {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
9124                {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
9125                {'call': 'prlimit64',
9126                 'reason': set([('mm_segment_t', 'seg')])},
9127                {'call': 'perf_event_open',
9128                 'reason': set([('mm_segment_t', 'seg')])},
9129                {'call': 'rt_sigaction',
9130                 'reason': set([('mm_segment_t', 'seg')])},
9131                {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
9132                {'call': 'getpriority',
9133                 'reason': set([('mm_segment_t', 'seg')])},
9134                {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
9135                {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
9136                {'call': 'get_robust_list',
9137                 'reason': set([('mm_segment_t', 'seg')])},
9138                {'call': 'mq_timedsend',
9139                 'reason': set([('mm_segment_t', 'seg')])},
9140                {'call': 'sched_getscheduler',
9141                 'reason': set([('mm_segment_t', 'seg')])},
9142                {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
9143                {'call': 'sched_getattr',
9144                 'reason': set([('mm_segment_t', 'seg')])},
9145                {'call': 'getrusage',
9146                 'reason': set([('mm_segment_t', 'seg')])},
9147                {'call': 'sched_setscheduler',
9148                 'reason': set([('mm_segment_t', 'seg')])},
9149                {'call': 'setitimer',
9150                 'reason': set([('mm_segment_t', 'seg')])},
9151                {'call': 'ioprio_get',
9152                 'reason': set([('mm_segment_t', 'seg')])},
9153                {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
9154                {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
9155                {'call': 'move_pages',
9156                 'reason': set([('mm_segment_t', 'seg')])},
9157                {'call': 'setpriority',
9158                 'reason': set([('mm_segment_t', 'seg')])},
9159                {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
9160                {'call': 'sched_getparam',
9161                 'reason': set([('mm_segment_t', 'seg')])}],
9162  'signalfd4': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
9163                {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
9164                {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
9165                {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
9166                {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
9167                {'call': 'readahead', 'reason': set([('fd', 'flags')])},
```



```
9168                {'call': 'getdents', 'reason': set([('fd', 'flags')])},
9169                {'call': 'writev', 'reason': set([('fd', 'flags')])},
9170                {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
9171                {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
9172                {'call': 'pread64', 'reason': set([('fd', 'flags')])},
9173                {'call': 'read', 'reason': set([('fd', 'flags')])},
9174                {'call': 'fchown', 'reason': set([('fd', 'flags')])},
9175                {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
9176                {'call': 'utime', 'reason': set([('fd', 'flags')])},
9177                {'call': 'fsync', 'reason': set([('fd', 'flags')])},
9178                {'call': 'bpf', 'reason': set([('fd', 'flags')])},
9179                {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
9180                {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
9181                {'call': 'sendto', 'reason': set([('fd', 'flags')])},
9182                {'call': 'tee', 'reason': set([('fd', 'flags')])},
9183                {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
9184                {'call': 'lseek', 'reason': set([('fd', 'flags')])},
9185                {'call': 'connect', 'reason': set([('fd', 'flags')])},
9186                {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
9187                {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
9188                {'call': 'flock', 'reason': set([('fd', 'flags')])},
9189                {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
9190                {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
9191                {'call': 'accept4', 'reason': set([('fd', 'flags')])},
9192                {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
9193                {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
9194                {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
9195                {'call': 'inotify_add_watch',
9196                 'reason': set([('fd', 'flags')])},
9197                {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
9198                {'call': 'splice', 'reason': set([('fd', 'flags')])},
9199                {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
9200                {'call': 'preadv', 'reason': set([('fd', 'flags')])},
9201                {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
9202                {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
9203                {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
9204                {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
9205                {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
9206                {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
9207                {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
9208                {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
9209                {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
9210                {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
9211                {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
9212                {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
9213                {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
9214                {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
9215                {'call': 'listen', 'reason': set([('fd', 'flags')])},
9216                {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
9217                {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
9218                {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
9219                {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
9220                {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
9221                {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
9222                {'call': 'llseek', 'reason': set([('fd', 'flags')])},
9223                {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
9224                {'call': 'readv', 'reason': set([('fd', 'flags')])},
9225                {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
9226                {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
9227                {'call': 'write', 'reason': set([('fd', 'flags')])},
9228                {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
9229                {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
9230                {'call': 'bind', 'reason': set([('fd', 'flags')])},
9231                {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
9232                {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
9233  'splice': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
```

```
9234            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
9235            {'call': 'fadvise64_64',
9236             'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
9237            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
9238            {'call': 'swapoff',
9239             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9240            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
9241            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
9242            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
9243            {'call': 'writev', 'reason': set([('fd', 'flags')])},
9244            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
9245            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
9246            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
9247            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
9248            {'call': 'memfd_create',
9249             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9250            {'call': 'remap_file_pages',
9251             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9252            {'call': 'dup3',
9253             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9254            {'call': 'read', 'reason': set([('fd', 'flags')])},
9255            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
9256            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
9257            {'call': 'utime', 'reason': set([('fd', 'flags')])},
9258            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
9259            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
9260            {'call': 'socketpair',
9261             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9262            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
9263            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
9264            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
9265            {'call': 'epoll_create1',
9266             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9267            {'call': 'tee', 'reason': set([('fd', 'flags')])},
9268            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
9269            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
9270            {'call': 'connect', 'reason': set([('fd', 'flags')])},
9271            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
9272            {'call': 'epoll_ctl',
9273             'reason': set([('fd', 'flags'),
9274                            ('file', 'f_flags'),
9275                            ('file', 'f_mode')])},
9276            {'call': 'flock',
9277             'reason': set([('fd', 'flags'),
9278                            ('file', 'f_flags'),
9279                            ('file', 'f_mode')])},
9280            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
9281            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
9282            {'call': 'openat',
9283             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9284            {'call': 'uselib',
9285             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9286            {'call': 'accept4',
9287             'reason': set([('fd', 'flags'),
9288                            ('file', 'f_flags'),
9289                            ('file', 'f_mode')])},
9290            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
9291            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
9292            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
9293            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
9294            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
9295            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
9296            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
9297            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
9298            {'call': 'shmat',
9299             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
```



```
9300            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
9301            {'call': 'socket',
9302             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9303            {'call': 'pipe2',
9304             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9305            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
9306            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
9307            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
9308            {'call': 'perf_event_open',
9309             'reason': set([('fd', 'flags'),
9310                            ('file', 'f_flags'),
9311                            ('file', 'f_mode')])},
9312            {'call': 'shmdt',
9313             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9314            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
9315            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
9316            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
9317            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
9318            {'call': 'acct',
9319             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9320            {'call': 'open',
9321             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9322            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
9323            {'call': 'mq_getsetattr',
9324             'reason': set([('fd', 'flags'), ('file', 'f_flags')])},
9325            {'call': 'dup',
9326             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9327            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
9328            {'call': 'setns',
9329             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9330            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
9331            {'call': 'listen', 'reason': set([('fd', 'flags')])},
9332            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
9333            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
9334            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
9335            {'call': 'shmctl',
9336             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9337            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
9338            {'call': 'swapon',
9339             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9340            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
9341            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
9342            {'call': 'eventfd2',
9343             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9344            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
9345            {'call': 'mmap_pgoff',
9346             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9347            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
9348            {'call': 'readv', 'reason': set([('fd', 'flags')])},
9349            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
9350            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
9351            {'call': 'write', 'reason': set([('fd', 'flags')])},
9352            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
9353            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
9354            {'call': 'mq_open',
9355             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9356            {'call': 'msync',
9357             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9358            {'call': 'open_by_handle_at',
9359             'reason': set([('file', 'f_flags'), ('file', 'f_mode')])},
9360            {'call': 'bind', 'reason': set([('fd', 'flags')])},
9361            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
9362            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
9363  'stat': [{'call': 'lstat',
9364            'reason': set([('__old_kernel_stat', 'st_ino'),
9365                           ('__old_kernel_stat', 'st_nlink')])},
```

```
9366          {'call': 'fstat',
9367           'reason': set([('__old_kernel_stat', 'st_ino'),
9368                          ('__old_kernel_stat', 'st_nlink')])}],
9369  'statfs': [{'call': 'ustat',
9370              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
9371             {'call': 'fstatfs',
9372              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
9373             {'call': 'fstatfs64',
9374              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])},
9375             {'call': 'statfs64',
9376              'reason': set([('kstatfs', 'f_ffree'), ('kstatfs', 'f_files')])}],
9377  'statfs64': [{'call': 'ustat',
9378                'reason': set([('kstatfs', 'f_ffree'),
9379                               ('kstatfs', 'f_files')])},
9380               {'call': 'fstatfs',
9381                'reason': set([('kstatfs', 'f_ffree'),
9382                               ('kstatfs', 'f_files')])},
9383               {'call': 'statfs',
9384                'reason': set([('kstatfs', 'f_ffree'),
9385                               ('kstatfs', 'f_files')])},
9386               {'call': 'fstatfs64',
9387                'reason': set([('kstatfs', 'f_ffree'),
9388                               ('kstatfs', 'f_files')])}],
9389  'swapoff': [{'call': 'swapon',
9390               'reason': set([('page', 'private'),
9391                              ('swap_info_struct', 'flags'),
9392                              ('swap_info_struct', 'inuse_pages'),
9393                              ('swap_info_struct', 'pages'),
9394                              ('swap_info_struct', 'prio'),
9395                              ('swap_info_struct', 'swap_map')])}],
9396  'swapon': [{'call': 'fadvise64_64', 'reason': set([('inode', 'i_flags')])},
9397             {'call': 'mq_unlink', 'reason': set([('inode', 'i_flags')])},
9398             {'call': 'swapoff',
9399              'reason': set([('inode', 'i_flags'),
9400                             ('swap_info_struct', 'flags'),
9401                             ('swap_info_struct', 'type')])},
9402             {'call': 'fchmod', 'reason': set([('inode', 'i_flags')])},
9403             {'call': 'memfd_create', 'reason': set([('inode', 'i_flags')])},
9404             {'call': 'readlinkat', 'reason': set([('inode', 'i_flags')])},
9405             {'call': 'fchown', 'reason': set([('inode', 'i_flags')])},
9406             {'call': 'mq_timedreceive',
9407              'reason': set([('inode', 'i_flags')])},
9408             {'call': 'uselib', 'reason': set([('inode', 'i_flags')])},
9409             {'call': 'fchmodat', 'reason': set([('inode', 'i_flags')])},
9410             {'call': 'inotify_add_watch',
9411              'reason': set([('inode', 'i_flags')])},
9412             {'call': 'ftruncate', 'reason': set([('inode', 'i_flags')])},
9413             {'call': 'ioctl', 'reason': set([('inode', 'i_flags')])},
9414             {'call': 'linkat', 'reason': set([('inode', 'i_flags')])},
9415             {'call': 'unlink', 'reason': set([('inode', 'i_flags')])},
9416             {'call': 'mq_getsetattr', 'reason': set([('inode', 'i_flags')])},
9417             {'call': 'faccessat', 'reason': set([('inode', 'i_flags')])},
9418             {'call': 'mq_timedsend', 'reason': set([('inode', 'i_flags')])},
9419             {'call': 'fchownat', 'reason': set([('inode', 'i_flags')])},
9420             {'call': 'mq_notify', 'reason': set([('inode', 'i_flags')])},
9421             {'call': 'sendfile', 'reason': set([('inode', 'i_flags')])},
9422             {'call': 'unlinkat', 'reason': set([('inode', 'i_flags')])},
9423             {'call': 'sendfile64', 'reason': set([('inode', 'i_flags')])}],
9424  'symlinkat': [{'call': 'sysfs',
9425                 'reason': set([('filename', 'name'),
9426                                ('filename', 'refcnt')])},
9427                {'call': 'mq_unlink',
9428                 'reason': set([('filename', 'name'),
9429                                ('filename', 'refcnt')])},
9430                {'call': 'swapoff',
9431                 'reason': set([('filename', 'name'),
```



```
9432                                ('filename', 'refcnt')])},
9433               {'call': 'openat',
9434                'reason': set([('filename', 'name'),
9435                               ('filename', 'refcnt')])},
9436               {'call': 'uselib',
9437                'reason': set([('filename', 'name'),
9438                               ('filename', 'refcnt')])},
9439               {'call': 'renameat2',
9440                'reason': set([('filename', 'name'),
9441                               ('filename', 'refcnt')])},
9442               {'call': 'quotactl',
9443                'reason': set([('filename', 'name'),
9444                               ('filename', 'refcnt')])},
9445               {'call': 'acct',
9446                'reason': set([('filename', 'name'),
9447                               ('filename', 'refcnt')])},
9448               {'call': 'open',
9449                'reason': set([('filename', 'name'),
9450                               ('filename', 'refcnt')])},
9451               {'call': 'unlink',
9452                'reason': set([('filename', 'name'),
9453                               ('filename', 'refcnt')])},
9454               {'call': 'rmdir',
9455                'reason': set([('filename', 'name'),
9456                               ('filename', 'refcnt')])},
9457               {'call': 'swapon',
9458                'reason': set([('filename', 'name'),
9459                               ('filename', 'refcnt')])},
9460               {'call': 'mq_open',
9461                'reason': set([('filename', 'name'),
9462                               ('filename', 'refcnt')])},
9463               {'call': 'unlinkat',
9464                'reason': set([('filename', 'name'),
9465                               ('filename', 'refcnt')])}],
9466  'sync_file_range': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
9467                      {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
9468                      {'call': 'fadvise64_64',
9469                       'reason': set([('fd', 'flags')])},
9470                      {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
9471                      {'call': 'fremovexattr',
9472                       'reason': set([('fd', 'flags')])},
9473                      {'call': 'readahead', 'reason': set([('fd', 'flags')])},
9474                      {'call': 'getdents', 'reason': set([('fd', 'flags')])},
9475                      {'call': 'writev', 'reason': set([('fd', 'flags')])},
9476                      {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
9477                      {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
9478                      {'call': 'pread64', 'reason': set([('fd', 'flags')])},
9479                      {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
9480                      {'call': 'read', 'reason': set([('fd', 'flags')])},
9481                      {'call': 'fchown', 'reason': set([('fd', 'flags')])},
9482                      {'call': 'mq_timedreceive',
9483                       'reason': set([('fd', 'flags')])},
9484                      {'call': 'utime', 'reason': set([('fd', 'flags')])},
9485                      {'call': 'fsync', 'reason': set([('fd', 'flags')])},
9486                      {'call': 'bpf', 'reason': set([('fd', 'flags')])},
9487                      {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
9488                      {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
9489                      {'call': 'sendto', 'reason': set([('fd', 'flags')])},
9490                      {'call': 'tee', 'reason': set([('fd', 'flags')])},
9491                      {'call': 'lseek', 'reason': set([('fd', 'flags')])},
9492                      {'call': 'connect', 'reason': set([('fd', 'flags')])},
9493                      {'call': 'getsockname',
9494                       'reason': set([('fd', 'flags')])},
9495                      {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
9496                      {'call': 'flock', 'reason': set([('fd', 'flags')])},
9497                      {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
```

```
                    {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
                    {'call': 'accept4', 'reason': set([('fd', 'flags')])},
                    {'call': 'old_readdir',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'inotify_rm_watch',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
                    {'call': 'inotify_add_watch',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
                    {'call': 'splice', 'reason': set([('fd', 'flags')])},
                    {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
                    {'call': 'preadv', 'reason': set([('fd', 'flags')])},
                    {'call': 'getpeername',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
                    {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
                    {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
                    {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
                    {'call': 'perf_event_open',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'pwritev64v2',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'futimesat','reason': set([('fd', 'flags')])},
                    {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
                    {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
                    {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
                    {'call': 'mq_getsetattr',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
                    {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
                    {'call': 'listen', 'reason': set([('fd', 'flags')])},
                    {'call': 'copy_file_range',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'mq_timedsend',
                     'reason': set([('fd', 'flags')])},
                    {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
                    {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
                    {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
                    {'call': 'llseek', 'reason': set([('fd', 'flags')])},
                    {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
                    {'call': 'readv', 'reason': set([('fd', 'flags')])},
                    {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
                    {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
                    {'call': 'write', 'reason': set([('fd', 'flags')])},
                    {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
                    {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
                    {'call': 'bind', 'reason': set([('fd', 'flags')])},
                    {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
                    {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'syncfs': [{'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
            {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
            {'call': 'writev', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
            {'call': 'read', 'reason': set([('fd', 'flags')])},
            {'call': 'fchown', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
            {'call': 'utime', 'reason': set([('fd', 'flags')])},
```



```
            {'call': 'fsync', 'reason': set([('fd', 'flags')])},
            {'call': 'bpf', 'reason': set([('fd', 'flags')])},
            {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
            {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendto', 'reason': set([('fd', 'flags')])},
            {'call': 'tee', 'reason': set([('fd', 'flags')])},
            {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'lseek', 'reason': set([('fd', 'flags')])},
            {'call': 'connect', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
            {'call': 'flock', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
            {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
            {'call': 'accept4', 'reason': set([('fd', 'flags')])},
            {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
            {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
            {'call': 'splice', 'reason': set([('fd', 'flags')])},
            {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv', 'reason': set([('fd', 'flags')])},
            {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
            {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
            {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
            {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
            {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
            {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
            {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
            {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
            {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
            {'call': 'listen', 'reason': set([('fd', 'flags')])},
            {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
            {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
            {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
            {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
            {'call': 'llseek', 'reason': set([('fd', 'flags')])},
            {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
            {'call': 'readv', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
            {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
            {'call': 'write', 'reason': set([('fd', 'flags')])},
            {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
            {'call': 'bind', 'reason': set([('fd', 'flags')])},
            {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
            {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
 'sysfs': [{'call': 'mq_unlink', 'reason': set([('filename', 'name')])},
           {'call': 'swapoff', 'reason': set([('filename', 'name')])},
           {'call': 'openat', 'reason': set([('filename', 'name')])},
           {'call': 'uselib', 'reason': set([('filename', 'name')])},
           {'call': 'renameat2', 'reason': set([('filename', 'name')])},
           {'call': 'symlinkat', 'reason': set([('filename', 'name')])},
           {'call': 'quotactl', 'reason': set([('filename', 'name')])},
           {'call': 'acct', 'reason': set([('filename', 'name')])},
           {'call': 'open', 'reason': set([('filename', 'name')])},
           {'call': 'unlink', 'reason': set([('filename', 'name')])},
           {'call': 'rmdir', 'reason': set([('filename', 'name')])},
```

```
9630            {'call': 'swapon', 'reason': set([('filename', 'name')])},
9631            {'call': 'mq_open', 'reason': set([('filename', 'name')])},
9632            {'call': 'unlinkat', 'reason': set([('filename', 'name')])}],
9633   'sysinfo': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
9634               {'call': 'rt_sigtimedwait',
9635                'reason': set([('mm_segment_t', 'seg'),
9636                               ('timespec', 'tv_nsec')])},
9637               {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
9638               {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
9639               {'call': 'fadvise64_64',
9640                'reason': set([('timespec', 'tv_nsec')])},
9641               {'call': 'mq_unlink', 'reason': set([('timespec', 'tv_nsec')])},
9642               {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
9643               {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
9644               {'call': 'sched_getaffinity',
9645                'reason': set([('mm_segment_t', 'seg')])},
9646               {'call': 'sched_setparam',
9647                'reason': set([('mm_segment_t', 'seg')])},
9648               {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
9649               {'call': 'memfd_create',
9650                'reason': set([('timespec', 'tv_nsec')])},
9651               {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
9652               {'call': 'readlinkat', 'reason': set([('timespec', 'tv_nsec')])},
9653               {'call': 'io_getevents',
9654                'reason': set([('timespec', 'tv_nsec')])},
9655               {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
9656               {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
9657               {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
9658               {'call': 'mq_timedreceive',
9659                'reason': set([('mm_segment_t', 'seg'),
9660                               ('timespec', 'tv_nsec')])},
9661               {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
9662               {'call': 'utime', 'reason': set([('timespec', 'tv_nsec')])},
9663               {'call': 'sched_setaffinity',
9664                'reason': set([('mm_segment_t', 'seg')])},
9665               {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
9666               {'call': 'semtimedop',
9667                'reason': set([('mm_segment_t', 'seg'),
9668                               ('timespec', 'tv_nsec')])},
9669               {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
9670               {'call': 'settimeofday',
9671                'reason': set([('timespec', 'tv_nsec')])},
9672               {'call': 'sched_rr_get_interval',
9673                'reason': set([('mm_segment_t', 'seg'),
9674                               ('timespec', 'tv_nsec')])},
9675               {'call': 'timerfd_gettime',
9676                'reason': set([('timespec', 'tv_nsec')])},
9677               {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
9678               {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
9679               {'call': 'rt_sigprocmask',
9680                'reason': set([('mm_segment_t', 'seg')])},
9681               {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
9682               {'call': 'sigaltstack',
9683                'reason': set([('mm_segment_t', 'seg')])},
9684               {'call': 'sched_setattr',
9685                'reason': set([('mm_segment_t', 'seg')])},
9686               {'call': 'migrate_pages',
9687                'reason': set([('mm_segment_t', 'seg')])},
9688               {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
9689               {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
9690               {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
9691               {'call': 'inotify_add_watch',
9692                'reason': set([('timespec', 'tv_nsec')])},
9693               {'call': 'timer_settime',
9694                'reason': set([('timespec', 'tv_nsec')])},
9695               {'call': 'ftruncate', 'reason': set([('timespec', 'tv_nsec')])},
```



```
9696               {'call': 'timer_gettime',
9697                'reason': set([('timespec', 'tv_nsec')])},
9698               {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
9699               {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
9700               {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
9701               {'call': 'perf_event_open',
9702                'reason': set([('mm_segment_t', 'seg')])},
9703               {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
9704               {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
9705               {'call': 'rt_sigaction',
9706                'reason': set([('mm_segment_t', 'seg')])},
9707               {'call': 'futimesat', 'reason': set([('timespec', 'tv_nsec')])},
9708               {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
9709               {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
9710               {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
9711               {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
9712               {'call': 'getpriority',
9713                'reason': set([('mm_segment_t', 'seg')])},
9714               {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
9715               {'call': 'nanosleep', 'reason': set([('timespec', 'tv_nsec')])},
9716               {'call': 'mq_getsetattr',
9717                'reason': set([('timespec', 'tv_nsec')])},
9718               {'call': 'faccessat', 'reason': set([('timespec', 'tv_nsec')])},
9719               {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
9720               {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
9721               {'call': 'get_robust_list',
9722                'reason': set([('mm_segment_t', 'seg')])},
9723               {'call': 'mq_timedsend',
9724                'reason': set([('mm_segment_t', 'seg'),
9725                               ('timespec', 'tv_nsec')])},
9726               {'call': 'sched_getscheduler',
9727                'reason': set([('mm_segment_t', 'seg')])},
9728               {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
9729               {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
9730               {'call': 'epoll_wait', 'reason': set([('timespec', 'tv_nsec')])},
9731               {'call': 'sched_getattr',
9732                'reason': set([('mm_segment_t', 'seg')])},
9733               {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
9734               {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
9735               {'call': 'timerfd_settime',
9736                'reason': set([('timespec', 'tv_nsec')])},
9737               {'call': 'sched_setscheduler',
9738                'reason': set([('mm_segment_t', 'seg')])},
9739               {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
9740               {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
9741               {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
9742               {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
9743               {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
9744               {'call': 'setpriority',
9745                'reason': set([('mm_segment_t', 'seg')])},
9746               {'call': 'mq_notify', 'reason': set([('timespec', 'tv_nsec')])},
9747               {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
9748               {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
9749               {'call': 'clock_nanosleep',
9750                'reason': set([('timespec', 'tv_nsec')])},
9751               {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
9752               {'call': 'sched_getparam',
9753                'reason': set([('mm_segment_t', 'seg')])},
9754               {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
9755               {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
9756               {'call': 'sendfile64', 'reason': set([('timespec', 'tv_nsec')])},
9757               {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
9758   'syslog': [{'call': 'keyctl', 'reason': set([('mm_segment_t', 'seg')])},
9759              {'call': 'rt_sigtimedwait',
9760               'reason': set([('mm_segment_t', 'seg')])},
9761              {'call': 'iopl', 'reason': set([('mm_segment_t', 'seg')])},
```

```
            {'call': 'msgrcv', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'kill', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getaffinity',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setparam',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioprio_set', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getppid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioperm', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'mq_timedreceive',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'capget', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setaffinity',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'signal', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'semtimedop', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'umount', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_rr_get_interval',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'rt_sigprocmask',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setsid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sigaltstack', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setattr',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'migrate_pages',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getitimer', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setpgid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getsid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'prlimit64', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'perf_event_open',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'rt_sigaction',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getpgid', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getpriority', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sigaction', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setns', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'fork', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'get_robust_list',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'mq_timedsend',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getscheduler',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ptrace', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getattr',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'getrusage', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_setscheduler',
             'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setitimer', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'ioprio_get', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'vfork', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'prctl', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'move_pages', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'setpriority', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'clone', 'reason': set([('mm_segment_t', 'seg')])},
            {'call': 'sched_getparam',
             'reason': set([('mm_segment_t', 'seg')])}],
 'tee': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
         {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
         {'call': 'fadvise64_64',
          'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
         {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
```



```
         {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
         {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
         {'call': 'readahead', 'reason': set([('fd', 'flags')])},
         {'call': 'getdents', 'reason': set([('fd', 'flags')])},
         {'call': 'writev', 'reason': set([('fd', 'flags')])},
         {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
         {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
         {'call': 'pread64', 'reason': set([('fd', 'flags')])},
         {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
         {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
         {'call': 'remap_file_pages', 'reason': set([('file', 'f_mode')])},
         {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
         {'call': 'read', 'reason': set([('fd', 'flags')])},
         {'call': 'fchown', 'reason': set([('fd', 'flags')])},
         {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
         {'call': 'utime', 'reason': set([('fd', 'flags')])},
         {'call': 'fsync', 'reason': set([('fd', 'flags')])},
         {'call': 'bpf', 'reason': set([('fd', 'flags')])},
         {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
         {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
         {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
         {'call': 'sendto', 'reason': set([('fd', 'flags')])},
         {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
         {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
         {'call': 'lseek', 'reason': set([('fd', 'flags')])},
         {'call': 'connect', 'reason': set([('fd', 'flags')])},
         {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
         {'call': 'epoll_ctl',
          'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
         {'call': 'flock',
          'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
         {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
         {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
         {'call': 'openat', 'reason': set([('file', 'f_mode')])},
         {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
         {'call': 'accept4',
          'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
         {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
         {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
         {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
         {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
         {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
         {'call': 'splice', 'reason': set([('fd', 'flags')])},
         {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
         {'call': 'preadv', 'reason': set([('fd', 'flags')])},
         {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
         {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
         {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
         {'call': 'socket', 'reason': set([('file', 'f_mode')])},
         {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
         {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
         {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
         {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
         {'call': 'perf_event_open',
          'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
         {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
         {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
         {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
         {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
         {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
         {'call': 'acct', 'reason': set([('file', 'f_mode')])},
         {'call': 'open', 'reason': set([('file', 'f_mode')])},
         {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
         {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
         {'call': 'dup', 'reason': set([('file', 'f_mode')])},
         {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
```

```
9894          {'call': 'setns', 'reason': set([('file', 'f_mode')])},
9895          {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
9896          {'call': 'listen', 'reason': set([('fd', 'flags')])},
9897          {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
9898          {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
9899          {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
9900          {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
9901          {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
9902          {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
9903          {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
9904          {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
9905          {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
9906          {'call': 'llseek', 'reason': set([('fd', 'flags')])},
9907          {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
9908          {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
9909          {'call': 'readv', 'reason': set([('fd', 'flags')])},
9910          {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
9911          {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
9912          {'call': 'write', 'reason': set([('fd', 'flags')])},
9913          {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
9914          {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
9915          {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
9916          {'call': 'msync', 'reason': set([('file', 'f_mode')])},
9917          {'call': 'open_by_handle_at', 'reason': set([('file', 'f_mode')])},
9918          {'call': 'bind', 'reason': set([('fd', 'flags')])},
9919          {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
9920          {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
9921   'timer_create': [{'call': 'clock_getres',
9922                     'reason': set([('k_clock', 'timer_create')])},
9923                    {'call': 'timer_delete',
9924                     'reason': set([('k_clock', 'timer_create')])},
9925                    {'call': 'clock_gettime',
9926                     'reason': set([('k_clock', 'timer_create')])},
9927                    {'call': 'timer_settime',
9928                     'reason': set([('k_clock', 'timer_create')])},
9929                    {'call': 'timer_gettime',
9930                     'reason': set([('k_clock', 'timer_create')])},
9931                    {'call': 'clock_settime',
9932                     'reason': set([('k_clock', 'timer_create')])},
9933                    {'call': 'clock_nanosleep',
9934                     'reason': set([('k_clock', 'timer_create')])},
9935                    {'call': 'clock_adjtime',
9936                     'reason': set([('k_clock', 'timer_create')])}],
9937   'timer_delete': [{'call': 'clock_getres',
9938                     'reason': set([('k_clock', 'timer_del')])},
9939                    {'call': 'timer_create',
9940                     'reason': set([('k_clock', 'timer_del')])},
9941                    {'call': 'clock_gettime',
9942                     'reason': set([('k_clock', 'timer_del')])},
9943                    {'call': 'timer_settime',
9944                     'reason': set([('k_clock', 'timer_del')])},
9945                    {'call': 'timer_gettime',
9946                     'reason': set([('k_clock', 'timer_del')])},
9947                    {'call': 'clock_settime',
9948                     'reason': set([('k_clock', 'timer_del')])},
9949                    {'call': 'clock_nanosleep',
9950                     'reason': set([('k_clock', 'timer_del')])},
9951                    {'call': 'clock_adjtime',
9952                     'reason': set([('k_clock', 'timer_del')])}],
9953   'timer_gettime': [{'call': 'clock_getres',
9954                      'reason': set([('k_clock', 'timer_get')])},
9955                     {'call': 'timer_delete',
9956                      'reason': set([('k_clock', 'timer_get')])},
9957                     {'call': 'timer_create',
9958                      'reason': set([('k_clock', 'timer_get')])},
9959                     {'call': 'clock_gettime',
```



```
9960                      'reason': set([('k_clock', 'timer_get')])},
9961                     {'call': 'timer_settime',
9962                      'reason': set([('k_clock', 'timer_get')])},
9963                     {'call': 'clock_settime',
9964                      'reason': set([('k_clock', 'timer_get')])},
9965                     {'call': 'clock_nanosleep',
9966                      'reason': set([('k_clock', 'timer_get')])},
9967                     {'call': 'clock_adjtime',
9968                      'reason': set([('k_clock', 'timer_get')])}],
9969   'timer_settime': [{'call': 'clock_getres',
9970                      'reason': set([('k_clock', 'timer_set')])},
9971                     {'call': 'timer_delete',
9972                      'reason': set([('k_clock', 'timer_set')])},
9973                     {'call': 'timer_create',
9974                      'reason': set([('k_clock', 'timer_set')])},
9975                     {'call': 'clock_gettime',
9976                      'reason': set([('k_clock', 'timer_set')])},
9977                     {'call': 'timer_gettime',
9978                      'reason': set([('k_clock', 'timer_set')])},
9979                     {'call': 'clock_settime',
9980                      'reason': set([('k_clock', 'timer_set')])},
9981                     {'call': 'clock_nanosleep',
9982                      'reason': set([('k_clock', 'timer_set')])},
9983                     {'call': 'clock_adjtime',
9984                      'reason': set([('k_clock', 'timer_set')])}],
9985   'timerfd_create': [{'call': 'timerfd_gettime',
9986                       'reason': set([('timerfd_ctx', 'clockid')])},
9987                      {'call': 'timerfd_settime',
9988                       'reason': set([('timerfd_ctx', 'clockid')])}],
9989   'timerfd_gettime': [{'call': 'timerfd_settime',
9990                        'reason': set([('timerfd_ctx', 'expired'),
9991                                       ('timerfd_ctx', 'tintv')])},
9992                       {'call': 'timerfd_create',
9993                        'reason': set([('timerfd_ctx', 'expired'),
9994                                       ('timerfd_ctx', 'tintv')])}],
9995   'timerfd_settime': [{'call': 'timerfd_gettime',
9996                        'reason': set([('timerfd_ctx', 'expired'),
9997                                       ('timerfd_ctx', 'tintv')])},
9998                       {'call': 'timerfd_create',
9999                        'reason': set([('timerfd_ctx', 'expired'),
10000                                       ('timerfd_ctx', 'tintv')])}],
10001   'umount': [{'call': 'syncfs', 'reason': set([('super_block', 's_flags')])},
10002              {'call': 'keyctl', 'reason': set([('task_struct', 'flags')])},
10003              {'call': 'rt_sigtimedwait',
10004               'reason': set([('task_struct', 'flags')])},
10005              {'call': 'msgrcv', 'reason': set([('task_struct', 'flags')])},
10006              {'call': 'fadvise64_64',
10007               'reason': set([('super_block', 's_flags')])},
10008              {'call': 'mq_unlink', 'reason': set([('vfsmount', 'mnt_flags')])},
10009              {'call': 'kill', 'reason': set([('task_struct', 'flags')])},
10010              {'call': 'sched_getaffinity',
10011               'reason': set([('task_struct', 'flags')])},
10012              {'call': 'sched_setparam',
10013               'reason': set([('task_struct', 'flags')])},
10014              {'call': 'pivot_root',
10015               'reason': set([('vfsmount', 'mnt_flags')])},
10016              {'call': 'ioprio_set', 'reason': set([('task_struct', 'flags')])},
10017              {'call': 'getppid', 'reason': set([('task_struct', 'flags')])},
10018              {'call': 'mq_timedreceive',
10019               'reason': set([('task_struct', 'flags')])},
10020              {'call': 'capget', 'reason': set([('task_struct', 'flags')])},
10021              {'call': 'sched_setaffinity',
10022               'reason': set([('task_struct', 'flags')])},
10023              {'call': 'ustat', 'reason': set([('super_block', 's_flags')])},
10024              {'call': 'signal', 'reason': set([('task_struct', 'flags')])},
10025              {'call': 'setreuid', 'reason': set([('task_struct', 'flags')])},
```

```
10026             {'call': 'semtimedop', 'reason': set([('task_struct', 'flags')])},
10027             {'call': 'sched_rr_get_interval',
10028              'reason': set([('task_struct', 'flags')])},
10029             {'call': 'rt_sigprocmask',
10030              'reason': set([('task_struct', 'flags')])},
10031             {'call': 'setsid', 'reason': set([('task_struct', 'flags')])},
10032             {'call': 'sigaltstack',
10033              'reason': set([('task_struct', 'flags')])},
10034             {'call': 'sched_setattr',
10035              'reason': set([('task_struct', 'flags')])},
10036             {'call': 'migrate_pages',
10037              'reason': set([('task_struct', 'flags')])},
10038             {'call': 'getitimer', 'reason': set([('task_struct', 'flags')])},
10039             {'call': 'setpgid', 'reason': set([('task_struct', 'flags')])},
10040             {'call': 'getcwd', 'reason': set([('vfsmount', 'mnt_flags')])},
10041             {'call': 'getsid', 'reason': set([('task_struct', 'flags')])},
10042             {'call': 'prlimit64', 'reason': set([('task_struct', 'flags')])},
10043             {'call': 'perf_event_open',
10044              'reason': set([('task_struct', 'flags')])},
10045             {'call': 'quotactl', 'reason': set([('super_block', 's_flags')])},
10046             {'call': 'rt_sigaction',
10047              'reason': set([('task_struct', 'flags')])},
10048             {'call': 'getpgid', 'reason': set([('task_struct', 'flags')])},
10049             {'call': 'acct', 'reason': set([('vfsmount', 'mnt_flags')])},
10050             {'call': 'getpriority',
10051              'reason': set([('task_struct', 'flags')])},
10052             {'call': 'sigaction', 'reason': set([('task_struct', 'flags')])},
10053             {'call': 'setns', 'reason': set([('task_struct', 'flags')])},
10054             {'call': 'fork', 'reason': set([('task_struct', 'flags')])},
10055             {'call': 'get_robust_list',
10056              'reason': set([('task_struct', 'flags')])},
10057             {'call': 'mq_timedsend',
10058              'reason': set([('task_struct', 'flags')])},
10059             {'call': 'sched_getscheduler',
10060              'reason': set([('task_struct', 'flags')])},
10061             {'call': 'ptrace', 'reason': set([('task_struct', 'flags')])},
10062             {'call': 'swapon', 'reason': set([('super_block', 's_flags')])},
10063             {'call': 'sched_getattr',
10064              'reason': set([('task_struct', 'flags')])},
10065             {'call': 'getrusage', 'reason': set([('task_struct', 'flags')])},
10066             {'call': 'sched_setscheduler',
10067              'reason': set([('task_struct', 'flags')])},
10068             {'call': 'setresuid', 'reason': set([('task_struct', 'flags')])},
10069             {'call': 'setitimer', 'reason': set([('task_struct', 'flags')])},
10070             {'call': 'ioprio_get', 'reason': set([('task_struct', 'flags')])},
10071             {'call': 'vfork', 'reason': set([('task_struct', 'flags')])},
10072             {'call': 'setuid', 'reason': set([('task_struct', 'flags')])},
10073             {'call': 'prctl', 'reason': set([('task_struct', 'flags')])},
10074             {'call': 'move_pages', 'reason': set([('task_struct', 'flags')])},
10075             {'call': 'setpriority',
10076              'reason': set([('task_struct', 'flags')])},
10077             {'call': 'clone', 'reason': set([('task_struct', 'flags')])},
10078             {'call': 'mq_open', 'reason': set([('vfsmount', 'mnt_flags')])},
10079             {'call': 'sched_getparam',
10080              'reason': set([('task_struct', 'flags')])}],
10081  'uname': [{'call': 'keyctl',
10082             'reason': set([('task_struct', 'personality')])},
10083            {'call': 'rt_sigtimedwait',
10084             'reason': set([('task_struct', 'personality')])},
10085            {'call': 'msgrcv',
10086             'reason': set([('task_struct', 'personality')])},
10087            {'call': 'kill', 'reason': set([('task_struct', 'personality')])},
10088            {'call': 'sched_getaffinity',
10089             'reason': set([('task_struct', 'personality')])},
10090            {'call': 'sched_setparam',
10091             'reason': set([('task_struct', 'personality')])},
```



```
10092            {'call': 'ioprio_set',
10093             'reason': set([('task_struct', 'personality')])},
10094            {'call': 'personality',
10095             'reason': set([('task_struct', 'personality')])},
10096            {'call': 'getppid',
10097             'reason': set([('task_struct', 'personality')])},
10098            {'call': 'mq_timedreceive',
10099             'reason': set([('task_struct', 'personality')])},
10100            {'call': 'capget',
10101             'reason': set([('task_struct', 'personality')])},
10102            {'call': 'sched_setaffinity',
10103             'reason': set([('task_struct', 'personality')])},
10104            {'call': 'signal',
10105             'reason': set([('task_struct', 'personality')])},
10106            {'call': 'semtimedop',
10107             'reason': set([('task_struct', 'personality')])},
10108            {'call': 'umount',
10109             'reason': set([('task_struct', 'personality')])},
10110            {'call': 'sched_rr_get_interval',
10111             'reason': set([('task_struct', 'personality')])},
10112            {'call': 'rt_sigprocmask',
10113             'reason': set([('task_struct', 'personality')])},
10114            {'call': 'setsid',
10115             'reason': set([('task_struct', 'personality')])},
10116            {'call': 'sigaltstack',
10117             'reason': set([('task_struct', 'personality')])},
10118            {'call': 'sched_setattr',
10119             'reason': set([('task_struct', 'personality')])},
10120            {'call': 'migrate_pages',
10121             'reason': set([('task_struct', 'personality')])},
10122            {'call': 'getitimer',
10123             'reason': set([('task_struct', 'personality')])},
10124            {'call': 'setpgid',
10125             'reason': set([('task_struct', 'personality')])},
10126            {'call': 'getsid',
10127             'reason': set([('task_struct', 'personality')])},
10128            {'call': 'prlimit64',
10129             'reason': set([('task_struct', 'personality')])},
10130            {'call': 'perf_event_open',
10131             'reason': set([('task_struct', 'personality')])},
10132            {'call': 'rt_sigaction',
10133             'reason': set([('task_struct', 'personality')])},
10134            {'call': 'getpgid',
10135             'reason': set([('task_struct', 'personality')])},
10136            {'call': 'getpriority',
10137             'reason': set([('task_struct', 'personality')])},
10138            {'call': 'sigaction',
10139             'reason': set([('task_struct', 'personality')])},
10140            {'call': 'setns', 'reason': set([('task_struct', 'personality')])},
10141            {'call': 'fork', 'reason': set([('task_struct', 'personality')])},
10142            {'call': 'get_robust_list',
10143             'reason': set([('task_struct', 'personality')])},
10144            {'call': 'mq_timedsend',
10145             'reason': set([('task_struct', 'personality')])},
10146            {'call': 'sched_getscheduler',
10147             'reason': set([('task_struct', 'personality')])},
10148            {'call': 'ptrace',
10149             'reason': set([('task_struct', 'personality')])},
10150            {'call': 'sched_getattr',
10151             'reason': set([('task_struct', 'personality')])},
10152            {'call': 'getrusage',
10153             'reason': set([('task_struct', 'personality')])},
10154            {'call': 'sched_setscheduler',
10155             'reason': set([('task_struct', 'personality')])},
10156            {'call': 'setitimer',
10157             'reason': set([('task_struct', 'personality')])},
```

```
10158           {'call': 'ioprio_get',
10159            'reason': set([('task_struct', 'personality')])},
10160           {'call': 'vfork', 'reason': set([('task_struct', 'personality')])},
10161           {'call': 'prctl', 'reason': set([('task_struct', 'personality')])},
10162           {'call': 'move_pages',
10163            'reason': set([('task_struct', 'personality')])},
10164           {'call': 'setpriority',
10165            'reason': set([('task_struct', 'personality')])},
10166           {'call': 'clone', 'reason': set([('task_struct', 'personality')])},
10167           {'call': 'sched_getparam',
10168            'reason': set([('task_struct', 'personality')])}],
10169  'uselib': [{'call': 'syncfs', 'reason': set([('super_block', 's_iflags')])},
10170           {'call': 'fadvise64_64',
10171            'reason': set([('super_block', 's_iflags')])},
10172           {'call': 'mq_unlink', 'reason': set([('vfsmount', 'mnt_flags')])},
10173           {'call': 'pivot_root',
10174            'reason': set([('vfsmount', 'mnt_flags')])},
10175           {'call': 'ustat', 'reason': set([('super_block', 's_iflags')])},
10176           {'call': 'umount',
10177            'reason': set([('super_block', 's_iflags'),
10178                           ('vfsmount', 'mnt_flags')])},
10179           {'call': 'getcwd', 'reason': set([('vfsmount', 'mnt_flags')])},
10180           {'call': 'quotactl',
10181            'reason': set([('super_block', 's_iflags')])},
10182           {'call': 'acct', 'reason': set([('vfsmount', 'mnt_flags')])},
10183           {'call': 'swapon', 'reason': set([('super_block', 's_iflags')])},
10184           {'call': 'mq_open', 'reason': set([('vfsmount', 'mnt_flags')])}],
10185  'utime': [{'call': 'rt_sigtimedwait',
10186            'reason': set([('timespec', 'tv_nsec')])},
10187           {'call': 'fadvise64_64', 'reason': set([('timespec', 'tv_nsec')])},
10188           {'call': 'mq_unlink', 'reason': set([('timespec', 'tv_nsec')])},
10189           {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
10190           {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
10191           {'call': 'memfd_create', 'reason': set([('timespec', 'tv_nsec')])},
10192           {'call': 'readlinkat', 'reason': set([('timespec', 'tv_nsec')])},
10193           {'call': 'io_getevents', 'reason': set([('timespec', 'tv_nsec')])},
10194           {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
10195           {'call': 'mq_timedreceive',
10196            'reason': set([('timespec', 'tv_nsec')])},
10197           {'call': 'semtimedop', 'reason': set([('timespec', 'tv_nsec')])},
10198           {'call': 'settimeofday', 'reason': set([('timespec', 'tv_nsec')])},
10199           {'call': 'sched_rr_get_interval',
10200            'reason': set([('timespec', 'tv_nsec')])},
10201           {'call': 'timerfd_gettime',
10202            'reason': set([('timespec', 'tv_nsec')])},
10203           {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
10204           {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
10205           {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
10206           {'call': 'inotify_add_watch',
10207            'reason': set([('timespec', 'tv_nsec')])},
10208           {'call': 'timer_settime',
10209            'reason': set([('timespec', 'tv_nsec')])},
10210           {'call': 'ftruncate', 'reason': set([('timespec', 'tv_nsec')])},
10211           {'call': 'timer_gettime',
10212            'reason': set([('timespec', 'tv_nsec')])},
10213           {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
10214           {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
10215           {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
10216           {'call': 'futimesat', 'reason': set([('timespec', 'tv_nsec')])},
10217           {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
10218           {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
10219           {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
10220           {'call': 'nanosleep', 'reason': set([('timespec', 'tv_nsec')])},
10221           {'call': 'mq_getsetattr',
10222            'reason': set([('timespec', 'tv_nsec')])},
10223           {'call': 'faccessat', 'reason': set([('timespec', 'tv_nsec')])},
```



```
10224           {'call': 'mq_timedsend', 'reason': set([('timespec', 'tv_nsec')])},
10225           {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
10226           {'call': 'epoll_wait', 'reason': set([('timespec', 'tv_nsec')])},
10227           {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
10228           {'call': 'timerfd_settime',
10229            'reason': set([('timespec', 'tv_nsec')])},
10230           {'call': 'mq_notify', 'reason': set([('timespec', 'tv_nsec')])},
10231           {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
10232           {'call': 'clock_nanosleep',
10233            'reason': set([('timespec', 'tv_nsec')])},
10234           {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
10235           {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
10236           {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
10237           {'call': 'sendfile64', 'reason': set([('timespec', 'tv_nsec')])},
10238           {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
10239  'utimensat': [{'call': 'rt_sigtimedwait',
10240                 'reason': set([('timespec', 'tv_nsec')])},
10241                {'call': 'fadvise64_64',
10242                 'reason': set([('timespec', 'tv_nsec')])},
10243                {'call': 'mq_unlink',
10244                 'reason': set([('timespec', 'tv_nsec')])},
10245                {'call': 'swapoff', 'reason': set([('timespec', 'tv_nsec')])},
10246                {'call': 'fchmod', 'reason': set([('timespec', 'tv_nsec')])},
10247                {'call': 'memfd_create',
10248                 'reason': set([('timespec', 'tv_nsec')])},
10249                {'call': 'readlinkat',
10250                 'reason': set([('timespec', 'tv_nsec')])},
10251                {'call': 'io_getevents',
10252                 'reason': set([('timespec', 'tv_nsec')])},
10253                {'call': 'fchown', 'reason': set([('timespec', 'tv_nsec')])},
10254                {'call': 'mq_timedreceive',
10255                 'reason': set([('timespec', 'tv_nsec')])},
10256                {'call': 'utime', 'reason': set([('timespec', 'tv_nsec')])},
10257                {'call': 'semtimedop',
10258                 'reason': set([('timespec', 'tv_nsec')])},
10259                {'call': 'settimeofday',
10260                 'reason': set([('timespec', 'tv_nsec')])},
10261                {'call': 'sched_rr_get_interval',
10262                 'reason': set([('timespec', 'tv_nsec')])},
10263                {'call': 'timerfd_gettime',
10264                 'reason': set([('timespec', 'tv_nsec')])},
10265                {'call': 'pselect6', 'reason': set([('timespec', 'tv_nsec')])},
10266                {'call': 'uselib', 'reason': set([('timespec', 'tv_nsec')])},
10267                {'call': 'fchmodat', 'reason': set([('timespec', 'tv_nsec')])},
10268                {'call': 'inotify_add_watch',
10269                 'reason': set([('timespec', 'tv_nsec')])},
10270                {'call': 'timer_settime',
10271                 'reason': set([('timespec', 'tv_nsec')])},
10272                {'call': 'ftruncate',
10273                 'reason': set([('timespec', 'tv_nsec')])},
10274                {'call': 'timer_gettime',
10275                 'reason': set([('timespec', 'tv_nsec')])},
10276                {'call': 'ioctl', 'reason': set([('timespec', 'tv_nsec')])},
10277                {'call': 'linkat', 'reason': set([('timespec', 'tv_nsec')])},
10278                {'call': 'stime', 'reason': set([('timespec', 'tv_nsec')])},
10279                {'call': 'futimesat',
10280                 'reason': set([('timespec', 'tv_nsec')])},
10281                {'call': 'poll', 'reason': set([('timespec', 'tv_nsec')])},
10282                {'call': 'select', 'reason': set([('timespec', 'tv_nsec')])},
10283                {'call': 'unlink', 'reason': set([('timespec', 'tv_nsec')])},
10284                {'call': 'nanosleep',
10285                 'reason': set([('timespec', 'tv_nsec')])},
10286                {'call': 'mq_getsetattr',
10287                 'reason': set([('timespec', 'tv_nsec')])},
10288                {'call': 'faccessat',
10289                 'reason': set([('timespec', 'tv_nsec')])},
```

```
10290                {'call': 'mq_timedsend',
10291                 'reason': set([('timespec', 'tv_nsec')])},
10292                {'call': 'swapon', 'reason': set([('timespec', 'tv_nsec')])},
10293                {'call': 'epoll_wait',
10294                 'reason': set([('timespec', 'tv_nsec')])},
10295                {'call': 'fchownat', 'reason': set([('timespec', 'tv_nsec')])},
10296                {'call': 'timerfd_settime',
10297                 'reason': set([('timespec', 'tv_nsec')])},
10298                {'call': 'mq_notify',
10299                 'reason': set([('timespec', 'tv_nsec')])},
10300                {'call': 'sendfile', 'reason': set([('timespec', 'tv_nsec')])},
10301                {'call': 'clock_nanosleep',
10302                 'reason': set([('timespec', 'tv_nsec')])},
10303                {'call': 'unlinkat', 'reason': set([('timespec', 'tv_nsec')])},
10304                {'call': 'futex', 'reason': set([('timespec', 'tv_nsec')])},
10305                {'call': 'recvmmsg', 'reason': set([('timespec', 'tv_nsec')])},
10306                {'call': 'sendfile64',
10307                 'reason': set([('timespec', 'tv_nsec')])},
10308                {'call': 'ppoll', 'reason': set([('timespec', 'tv_nsec')])}],
10309  'vmsplice': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
10310               {'call': 'fadvise64_64',
10311                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10312               {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
10313               {'call': 'swapoff', 'reason': set([('file', 'f_mode')])},
10314               {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
10315               {'call': 'readahead', 'reason': set([('fd', 'flags')])},
10316               {'call': 'getdents', 'reason': set([('fd', 'flags')])},
10317               {'call': 'writev', 'reason': set([('fd', 'flags')])},
10318               {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
10319               {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
10320               {'call': 'pread64', 'reason': set([('fd', 'flags')])},
10321               {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
10322               {'call': 'memfd_create', 'reason': set([('file', 'f_mode')])},
10323               {'call': 'remap_file_pages',
10324                'reason': set([('file', 'f_mode')])},
10325               {'call': 'dup3', 'reason': set([('file', 'f_mode')])},
10326               {'call': 'read', 'reason': set([('fd', 'flags')])},
10327               {'call': 'fchown', 'reason': set([('fd', 'flags')])},
10328               {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
10329               {'call': 'utime', 'reason': set([('fd', 'flags')])},
10330               {'call': 'fsync', 'reason': set([('fd', 'flags')])},
10331               {'call': 'bpf', 'reason': set([('fd', 'flags')])},
10332               {'call': 'socketpair', 'reason': set([('file', 'f_mode')])},
10333               {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
10334               {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
10335               {'call': 'sendto', 'reason': set([('fd', 'flags')])},
10336               {'call': 'epoll_create1', 'reason': set([('file', 'f_mode')])},
10337               {'call': 'tee', 'reason': set([('fd', 'flags')])},
10338               {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
10339               {'call': 'lseek', 'reason': set([('fd', 'flags')])},
10340               {'call': 'connect', 'reason': set([('fd', 'flags')])},
10341               {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
10342               {'call': 'epoll_ctl',
10343                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10344               {'call': 'flock',
10345                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10346               {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
10347               {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
10348               {'call': 'openat', 'reason': set([('file', 'f_mode')])},
10349               {'call': 'uselib', 'reason': set([('file', 'f_mode')])},
10350               {'call': 'accept4',
10351                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10352               {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
10353               {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
10354               {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
10355               {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
```



```
10356               {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
10357               {'call': 'splice', 'reason': set([('fd', 'flags')])},
10358               {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
10359               {'call': 'preadv', 'reason': set([('fd', 'flags')])},
10360               {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
10361               {'call': 'shmat', 'reason': set([('file', 'f_mode')])},
10362               {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
10363               {'call': 'socket', 'reason': set([('file', 'f_mode')])},
10364               {'call': 'pipe2', 'reason': set([('file', 'f_mode')])},
10365               {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
10366               {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
10367               {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
10368               {'call': 'perf_event_open',
10369                'reason': set([('fd', 'flags'), ('file', 'f_mode')])},
10370               {'call': 'shmdt', 'reason': set([('file', 'f_mode')])},
10371               {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
10372               {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
10373               {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
10374               {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
10375               {'call': 'acct', 'reason': set([('file', 'f_mode')])},
10376               {'call': 'open', 'reason': set([('file', 'f_mode')])},
10377               {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
10378               {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
10379               {'call': 'dup', 'reason': set([('file', 'f_mode')])},
10380               {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
10381               {'call': 'setns', 'reason': set([('file', 'f_mode')])},
10382               {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
10383               {'call': 'listen', 'reason': set([('fd', 'flags')])},
10384               {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
10385               {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
10386               {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
10387               {'call': 'shmctl', 'reason': set([('file', 'f_mode')])},
10388               {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
10389               {'call': 'swapon', 'reason': set([('file', 'f_mode')])},
10390               {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
10391               {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
10392               {'call': 'eventfd2', 'reason': set([('file', 'f_mode')])},
10393               {'call': 'llseek', 'reason': set([('fd', 'flags')])},
10394               {'call': 'mmap_pgoff', 'reason': set([('file', 'f_mode')])},
10395               {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
10396               {'call': 'readv', 'reason': set([('fd', 'flags')])},
10397               {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
10398               {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
10399               {'call': 'write', 'reason': set([('fd', 'flags')])},
10400               {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
10401               {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
10402               {'call': 'mq_open', 'reason': set([('file', 'f_mode')])},
10403               {'call': 'msync', 'reason': set([('file', 'f_mode')])},
10404               {'call': 'open_by_handle_at',
10405                'reason': set([('file', 'f_mode')])},
10406               {'call': 'bind', 'reason': set([('fd', 'flags')])},
10407               {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
10408               {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}],
10409  'write': [{'call': 'syncfs', 'reason': set([('fd', 'flags')])},
10410            {'call': 'vmsplice', 'reason': set([('fd', 'flags')])},
10411            {'call': 'fadvise64_64', 'reason': set([('fd', 'flags')])},
10412            {'call': 'pwritev64', 'reason': set([('fd', 'flags')])},
10413            {'call': 'fremovexattr', 'reason': set([('fd', 'flags')])},
10414            {'call': 'readahead', 'reason': set([('fd', 'flags')])},
10415            {'call': 'getdents', 'reason': set([('fd', 'flags')])},
10416            {'call': 'writev', 'reason': set([('fd', 'flags')])},
10417            {'call': 'preadv64', 'reason': set([('fd', 'flags')])},
10418            {'call': 'fchmod', 'reason': set([('fd', 'flags')])},
10419            {'call': 'pread64', 'reason': set([('fd', 'flags')])},
10420            {'call': 'signalfd4', 'reason': set([('fd', 'flags')])},
10421            {'call': 'read', 'reason': set([('fd', 'flags')])},
```

```
10422             {'call': 'fchown', 'reason': set([('fd', 'flags')])},
10423             {'call': 'mq_timedreceive', 'reason': set([('fd', 'flags')])},
10424             {'call': 'utime', 'reason': set([('fd', 'flags')])},
10425             {'call': 'fsync', 'reason': set([('fd', 'flags')])},
10426             {'call': 'bpf', 'reason': set([('fd', 'flags')])},
10427             {'call': 'recvfrom', 'reason': set([('fd', 'flags')])},
10428             {'call': 'fsetxattr', 'reason': set([('fd', 'flags')])},
10429             {'call': 'sendto', 'reason': set([('fd', 'flags')])},
10430             {'call': 'tee', 'reason': set([('fd', 'flags')])},
10431             {'call': 'sync_file_range', 'reason': set([('fd', 'flags')])},
10432             {'call': 'lseek', 'reason': set([('fd', 'flags')])},
10433             {'call': 'connect', 'reason': set([('fd', 'flags')])},
10434             {'call': 'getsockname', 'reason': set([('fd', 'flags')])},
10435             {'call': 'epoll_ctl', 'reason': set([('fd', 'flags')])},
10436             {'call': 'flock', 'reason': set([('fd', 'flags')])},
10437             {'call': 'pwritev', 'reason': set([('fd', 'flags')])},
10438             {'call': 'fchdir', 'reason': set([('fd', 'flags')])},
10439             {'call': 'accept4', 'reason': set([('fd', 'flags')])},
10440             {'call': 'old_readdir', 'reason': set([('fd', 'flags')])},
10441             {'call': 'inotify_rm_watch', 'reason': set([('fd', 'flags')])},
10442             {'call': 'utimensat', 'reason': set([('fd', 'flags')])},
10443             {'call': 'inotify_add_watch', 'reason': set([('fd', 'flags')])},
10444             {'call': 'preadv2', 'reason': set([('fd', 'flags')])},
10445             {'call': 'splice', 'reason': set([('fd', 'flags')])},
10446             {'call': 'ftruncate', 'reason': set([('fd', 'flags')])},
10447             {'call': 'preadv', 'reason': set([('fd', 'flags')])},
10448             {'call': 'getpeername', 'reason': set([('fd', 'flags')])},
10449             {'call': 'setsockopt', 'reason': set([('fd', 'flags')])},
10450             {'call': 'fcntl', 'reason': set([('fd', 'flags')])},
10451             {'call': 'ioctl', 'reason': set([('fd', 'flags')])},
10452             {'call': 'pwrite64', 'reason': set([('fd', 'flags')])},
10453             {'call': 'perf_event_open', 'reason': set([('fd', 'flags')])},
10454             {'call': 'pwritev64v2', 'reason': set([('fd', 'flags')])},
10455             {'call': 'futimesat', 'reason': set([('fd', 'flags')])},
10456             {'call': 'pwritev2', 'reason': set([('fd', 'flags')])},
10457             {'call': 'shutdown', 'reason': set([('fd', 'flags')])},
10458             {'call': 'getsockopt', 'reason': set([('fd', 'flags')])},
10459             {'call': 'mq_getsetattr', 'reason': set([('fd', 'flags')])},
10460             {'call': 'fdatasync', 'reason': set([('fd', 'flags')])},
10461             {'call': 'getdents64', 'reason': set([('fd', 'flags')])},
10462             {'call': 'listen', 'reason': set([('fd', 'flags')])},
10463             {'call': 'copy_file_range', 'reason': set([('fd', 'flags')])},
10464             {'call': 'mq_timedsend', 'reason': set([('fd', 'flags')])},
10465             {'call': 'fgetxattr', 'reason': set([('fd', 'flags')])},
10466             {'call': 'fcntl64', 'reason': set([('fd', 'flags')])},
10467             {'call': 'fallocate', 'reason': set([('fd', 'flags')])},
10468             {'call': 'epoll_wait', 'reason': set([('fd', 'flags')])},
10469             {'call': 'llseek', 'reason': set([('fd', 'flags')])},
10470             {'call': 'preadv64v2', 'reason': set([('fd', 'flags')])},
10471             {'call': 'readv', 'reason': set([('fd', 'flags')])},
10472             {'call': 'fstatfs', 'reason': set([('fd', 'flags')])},
10473             {'call': 'fstatfs64', 'reason': set([('fd', 'flags')])},
10474             {'call': 'mq_notify', 'reason': set([('fd', 'flags')])},
10475             {'call': 'sendfile', 'reason': set([('fd', 'flags')])},
10476             {'call': 'bind', 'reason': set([('fd', 'flags')])},
10477             {'call': 'flistxattr', 'reason': set([('fd', 'flags')])},
10478             {'call': 'sendfile64', 'reason': set([('fd', 'flags')])}]}
```

```
**********************************************************
    1442 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/kchecker
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

function usage {
    echo "Usage:  $0 [--sparse][--valgrind][--debug] path/to/file.c"
    exit 1
}

SCRIPT_DIR=$(dirname $0)
if [ -e $SCRIPT_DIR/../smatch ] ; then
    CMD=$SCRIPT_DIR/../smatch
elif which smatch | grep smatch > /dev/null ; then
    CMD=smatch
else
    echo "Smatch binary not found."
    exit 1
fi

while true ; do
    if [[ "$1" == "--sparse" ]] ; then
        CMD="sparse"
        shift
    elif [[ "$1" == "--test-parsing" ]] ; then
        CMD="$SCRIPT_DIR/../test-parsing -no-lineno"
        shift
    elif echo "$1" | grep -q -- "--outfile=" ; then
        outfile=$(echo "$1" | cut -d '=' -f 2)
        if [ "outfile" != "" ] ; then
            POST="$POST > $outfile"
        fi
        shift
    elif [[ "$1" == "--valgrind" ]] ; then
        PRE="valgrind"
        shift
    elif [[ "$1" == "--endian" ]] ; then
        ENDIAN="CF=-D__CHECK_ENDIAN__"
        shift
    elif [[ "$1" == "" ]] ; then
        break
    else
        if [[ "$1" == "--help" ]] ; then
                $CMD --help
                exit 1
        fi
        if echo $1 | grep -q ^- ; then
                POST="$1 $POST"
        else
                break
        fi
        shift
    fi
done
if echo $CMD | grep -q smatch ; then
    POST="--project=kernel --succeed $POST"
fi

cname=$1
cname=$(echo ${cname/.o/.c})
if [[ "$cname" == "" ]] ; then
    usage
fi
```



```
if ! test -e $cname ; then
    usage
fi

oname=$(echo ${cname/.c/.o})
if ! echo $oname | grep -q .o$ && ! echo $oname | grep -q /$ ; then
    usage
fi
if  echo $oname | grep -q .o$ ; then
    rm -f $oname
fi

make C=2 $ENDIAN CHECK="$PRE $CMD $POST" $oname
```

```
**********************************************************
    2142 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/kpatch.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 #!/bin/bash -e

  3 TMP_DIR=/tmp

  5 help()
  6 {
  7     echo "Usage: $0 [--no-compile|--ammend] <filename>"
  8     echo "You must be at the base of the kernel tree to run this."
  9     exit 1
 10 }

 12 continue_yn()
 13 {
 14     echo -n "Do you want to fix these issues now? "
 15     read ans
 16     if ! echo $ans | grep -iq ^n ; then
 17         exit 1;
 18     fi
 19 }

 21 qc()
 22 {
 23     local msg=$1
 24     local ans

 26     echo -n "$msg:  "
 27     read ans
 28     if ! echo $ans | grep -qi ^y ; then
 29         exit 1
 30     fi
 31 }

 33 NO_COMPILE=false
 34 AMEND=""

 36 while true ; do
 37     if [[ "$1" == "--no-compile" ]] ; then
 38         NO_COMPILE=true
 39         shift
 40     elif [[ "$1" == "--ammend" ]] ; then
 41         AMEND="--amend"
 42         shift
 43     else
 44         break
 45     fi
 46 done

 48 if [ ! -f $1 ] ; then
 49     help
 50 fi

 52 fullname=$1
 53 filename=$(basename $fullname)
 54 oname=$(echo ${fullname/.c/.o})

 56 MAIL_FILE=$TMP_DIR/${filename}.msg

 58 echo "QC checklist"
 59 qc "Have you handled all the errors properly?"
 60 if git diff $fullname | grep ^+ | grep -qi alloc ; then
```



```
 61     qc "Have you freed all your mallocs?"
 62 fi
 63 if git diff $fullname | grep ^+ | grep -qi alloc ; then
 64     qc "Have you check all your mallocs for NULL returns?"
 65 fi

 67 if [ "$NO_COMPILE" != "true" ] ; then
 68     kchecker --spammy $fullname
 69     kchecker --sparse --endian $fullname
 70 #    rm $oname
 71 #    make C=1 CHECK="scripts/coccicheck" $oname
 72 fi

 74 grepmail $fullname ~/var/mail/sent* | grep -i ^subject || echo -n ""
 75 qc "Looks OK?"

 77 git log --oneline $fullname | head -n 10
 78 echo "Copy and paste one of these subjects?"
 79 read unused

 81 git add $fullname
 82 git commit --signoff $AMEND

 84 to_addr=$(./scripts/get_maintainer.pl -f --noroles --norolestats $fullname | hea
 85 cc_addr=$(./scripts/get_maintainer.pl -f --noroles --norolestats $fullname | tai
 86     perl -ne 's/\n$/, /; print')
 87 cc_addr="$cc_addr, kernel-janitors@vger.kernel.org"

 89 echo -n "To:  "  > $MAIL_FILE
 90 echo "$to_addr" >> $MAIL_FILE
 91 echo -n "CC:  " >> $MAIL_FILE
 92 echo "$cc_addr" >> $MAIL_FILE
 93 echo "X-Mailer: git-send-email haha only kidding" >> $MAIL_FILE

 95 git format-patch HEAD^ --stdout >> $MAIL_FILE

 97 ./scripts/checkpatch.pl $MAIL_FILE || continue_yn

 99 echo "Press ENTER to continue"
100 read unused

102 mutt -H $MAIL_FILE
```

```
**********************************************************
     867 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/new_bugs.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

new=$1
old=$2

if [ "$old" = "" ] ; then
    echo "usage:  $0 <new file> <old file>"
    exit 1
fi

#
# If the $old and $new are very similar then we can
# filter out a lot of bug just by doing a diff.
#
# But the line numbers change quite frequently so
# really we only want to see if the line numbers
# have changed inside the function.
# The 42 in this message:
# file.c +123 some_func(42) warn: blah blah blah
#

IFS="
"
for err in $(diff -u $old $new | cut -b 2- | egrep '(warn|error|warning):') ; do

    # we are only interested in the last chunk.
    # "some_func(42) warn: blah blah blah"
    last=$(echo $err | cut -d ' ' -f 2-)

    # There are some error message which include a second
    # line number so we crudely chop that off.
    last=$(echo $last | sed -e 's/line .*//')

    if ! grep -Fq "$last" $old ; then
        echo $err
    fi
done
```

```
**********************************************************
     813 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/show_errs.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

context=6
while true ; do
    if [ "$1" = "-C" ] ; then
        shift
        context=$1
        shift
        continue
    fi
    break
done


file=$1
[ "$file" = "" ] && [ -e err-list ] && file=err-list
if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 [-C <lines>] [-b] [-k] <file with smatch messages>"
    echo "  -C <lines>:  Print <lines> of context"
    exit 1
fi

cat $file | while read line ; do
    code_file=$(echo "$line" | cut -d ':' -f 1)
    lineno=$(echo "$line" | cut -d ' ' -f 1 | cut -d ':' -f 2)
    echo "============================================================"
    echo "$line"
    echo "---"
    tail -n +$(($lineno - ($context - 1))) $code_file | head -n $(($context - 1)
    echo "------------------------------------------------------------"
    tail -n +${lineno} $code_file | head -n $context
done
```

```
**********************************************************
     696 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/show_ifs.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

context=1
if [ "$1" = "-C" ] ; then
    shift
    context=$1
    shift
fi

file=$1
if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 [-C <lines of context>] <file with smatch messages>"
    exit 1
fi

grep 'if();' $file | cut -d ' ' -f1 | while read loc; do
    code_file=$(echo $loc | cut -d ':' -f 1)
    line=$(echo $loc | cut -d ':' -f 2)
    echo "============================================================"
    echo $code_file $line
    tail -n +$(($line - ($context - 1))) $code_file | head -n $(($context - 1))
    if [[ $context -gt 1 ]] ; then
        echo "------------------------------------------------------------"
    fi
    tail -n +${line} $code_file | head -n $context
done
```

```
**********************************************************
    2025 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/show_unreachable.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

context=6
while true ; do
    if [ "$1" = "-C" ] ; then
        shift
        context=$1
        shift
        continue
    fi
    if [ "$1" = "-k" ] ; then
        shift
        mode=kernel
        continue
    fi
    if [ "$1" = "-b" ] ; then
        shift
        nobreak=yes
        continue
    fi
    break
done


file=$1
if [[ "$file" = "" ]] ; then
    echo "Usage:  $0 [-C <lines>] [-b] [-k] <file with smatch messages>"
    echo "  -C <lines>:  Print <lines> of context"
    echo "  -b         :  Ignore unreachable break statements"
    echo "  -k         :  Ignore some kernel defines"
    exit 1
fi

kernel_ignore_functions="DLM_ASSERT
BT_SI_SM_RETURN
BT_STATE_CHANGE
PARSE_ERROR1
PARSE_ERROR
CMDINSIZE
PROCESS_SYSTEM_PARAM
RETURN_STATUS
ar9170_regwrite_result
module_put_and_exit
SEG32
CASE_PIPExTRE
"

grep 'ignoring unreachable' $file | cut -d ' ' -f1 | while read loc; do
    code_file=$(echo $loc | cut -d ':' -f 1)
    line=$(echo $loc | cut -d ':' -f 2)

    if [ "$mode" = "kernel" ] ; then
        # BUG() is sometimes defined away on embedded systems
        if tail -n +$(($line - 1)) $code_file | head -n 1 | \
            egrep -qw '(BUG|BT_STATE_CHANGE)' ; then
            continue;
        fi
        skip=0
        line_txt=$(tail -n +$(($line)) $code_file | head -n 1)
        for func in $kernel_ignore_functions ; do
```



```
            if echo "$line_txt" | egrep -qw $func ; then
                skip=1
                break
            fi
        done
        if [ "$skip" == 1 ] ; then
            continue
        fi
    fi

    if [ "$nobreak" = "yes" ] ; then
        if tail -n +$(($line)) $code_file | head -n 1 | grep -qw 'break' ; then
            continue;
        fi

    fi
    echo "=========================================================="
    echo $code_file:$line
    tail -n +$(($line - ($context - 1))) $code_file | head -n $(($context - 1))
    echo "----------------------------------------------------------"
    tail -n +${line} $code_file | head -n $context
done
```

```
**********************************************************
     387 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/strip_whitespace.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #!/usr/bin/perl

   3 use strict;

   5 my $file = shift();
   6 open FILE, "<$file";
   7 my $txt = do { local $/;  <FILE> };

   9 # strip C99 comments
  10 $txt =~ s/\/\/.*//g;
  11 # strip newlines
  12 $txt =~ s/\n//g;
  13 # strip remaining comments
  14 $txt =~ s/\/\*.*?\*\///g;
  15 # strip tabs
  16 $txt =~ s/\t//g;
  17 # strip spaces
  18 $txt =~ s/ //g;
  19 # add newlines
  20 $txt =~ s/;/;\n/g;
  21 $txt =~ s/{/{\n/g;
  22 $txt =~ s/}/}\n/g;

  24 print "$txt\n";
```

```
**********************************************************
    1876 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/summarize_errs.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

print_help()
{
    echo "usage: $0 <warning file>"
    exit 1;
}

set_title()
{
    echo -ne "\033]0;$*\007"
    echo ========================================
    echo $*
    echo ----------------------------------------
}

cmd_help()
{
    echo "n - skips to next message"
    echo "f - skips to next file"
    echo "? - print this message again"
}

save_thoughts()
{
    echo "************"
    echo $sm_err
    echo -n "What do you think?:  "
    read ans
    if echo $ans | grep ^$ > /dev/null ; then
        continue
    fi

    #store the result
    echo $sm_err       >> summary
    echo $ans       >> summary
    echo ========== >> summary
}

if [ "$1" = "--new" ] ; then
    shift
    NEW=Y
fi

file=$1
if [ "$file" = "" ] ; then
    if [ -e err-list ] ; then
        file="err-list"
    else
        print_help
    fi
fi

TXT=$(cat $file | uniq -f 2)

IFS='
'
for sm_err in $TXT ; do
    file=$(echo $sm_err | cut -d ':' -f 1)
    line=$(echo $sm_err | cut -d ' ' -f 1 | cut -d ':' -f 2)
```



```
    if [ "$file" = "$skip_file" ] ; then
        continue
    fi
    skip_file=""

    last=$(echo $sm_err | cut -d ' ' -f 2-)
    last=$(echo $last | sed -e 's/line .*//')

    if [ "$NEW" = "Y" ] ; then
        if grep -F "$last" *summary* > /dev/null ; then
            echo "skipping $sm_err"
            continue
        fi
    fi

    set_title $sm_err

    #grep -A1 "$file $line" *summary* 2> /dev/null
    grep -A1 -F "$last" *summary* 2> /dev/null

    ans="?"
    while echo $ans | grep '?' > /dev/null ; do
        echo -n "[? for help]: "
        read ans
        if echo $ans | grep n > /dev/null ; then
            continue 2
        fi
        if echo $ans | grep f > /dev/null ; then
            skip_file=$file
            continue 2
        fi
        if echo $ans | grep '?' > /dev/null ; then
            cmd_help
        fi
    done

    # I have this in my .vimrc
    # map <C-j> :! echo $sm_err<CR>
    export sm_err

    vim $file +${line}

    save_thoughts
done
IFS=
```

```
**********************************************************
    1507 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/test_generic.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

NR_CPU=$(cat /proc/cpuinfo | grep ^processor | wc -l)
TARGET=""
WLOG="smatch_warns.txt"
LOG="smatch_compile.warns"
function usage {
    echo
    echo "Usage:  $0 [smatch options]"
    echo "Compiles the kernel with -j${NR_CPU}"
    echo " available options:"
    echo "      --endian          : enable endianess check"
    echo "      --target {TARGET} : specify build target, default: $TARGET"
    echo "      --log {FILE}      : Output compile log to file, default is: $LOG
    echo "      --wlog {FILE}     : Output warnigs to file, default is: $WLOG"
    echo "      --help            : Show this usage"
    exit 1
}


while true ; do
    if [[ "$1" == "--endian" ]] ; then
        ENDIAN="CF=-D__CHECK_ENDIAN__"
        shift
    elif [[ "$1" == "--target" ]] ; then
        shift
        TARGET="$1"
        shift
    elif [[ "$1" == "--log" ]] ; then
        shift
        LOG="$1"
        shift
    elif [[ "$1" == "--wlog" ]] ; then
        shift
        WLOG="$1"
        shift
    elif [[ "$1" == "--help" ]] ; then
        usage
    else
            break
    fi
done

SCRIPT_DIR=$(dirname $0)
if [ -e $SCRIPT_DIR/../smatch ] ; then
    cp $SCRIPT_DIR/../smatch $SCRIPT_DIR/../bak.smatch
    CMD=$SCRIPT_DIR/../bak.smatch
elif which smatch | grep smatch > /dev/null ; then
    CMD=smatch
else
    echo "Smatch binary not found."
    exit 1
fi

make clean
find -name \*.c.smatch -exec rm \{\} \;
make -j${NR_CPU} $ENDIAN -k CHECK="$CMD --file-output $*" \
        C=1 $TARGET 2>&1 | tee $LOG
find -name \*.c.smatch -exec cat \{\} \; -exec rm \{\} \; > $WLOG
```



```
echo "Done.  The warnings are saved to $WLOG"
```

```
**********************************************************
    1706 Fri Dec 21 15:00:35 2018
new/usr/src/tools/smatch/src/smatch_scripts/test_kernel.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

NR_CPU=$(cat /proc/cpuinfo | grep ^processor | wc -l)
TARGET="bzImage modules"
WLOG="smatch_warns.txt"
LOG="smatch_compile.warns"
function usage {
    echo
    echo "Usage:  $0 [smatch options]"
    echo "Compiles the kernel with -j${NR_CPU}"
    echo " available options:"
    echo "      --endian          : enable endianess check"
    echo "      --target {TARGET} : specify build target, default: $TARGET"
    echo "      --log {FILE}      : Output compile log to file, default is: $LOG
    echo "      --wlog {FILE}     : Output warnigs to file, default is: $WLOG"
    echo "      --help            : Show this usage"
    exit 1
}


while true ; do
    if [[ "$1" == "--endian" ]] ; then
        ENDIAN="CF=-D__CHECK_ENDIAN__"
        shift
    elif [[ "$1" == "--target" ]] ; then
        shift
        TARGET="$1"
        shift
    elif [[ "$1" == "--log" ]] ; then
        shift
        LOG="$1"
        shift
    elif [[ "$1" == "--wlog" ]] ; then
        shift
        WLOG="$1"
        shift
    elif [[ "$1" == "--help" ]] ; then
        usage
    else
            break
    fi
done

SCRIPT_DIR=$(dirname $0)
if [ -e $SCRIPT_DIR/../smatch ] ; then
    cp $SCRIPT_DIR/../smatch $SCRIPT_DIR/../bak.smatch
    CMD=$SCRIPT_DIR/../bak.smatch
elif which smatch | grep smatch > /dev/null ; then
    CMD=smatch
else
    echo "Smatch binary not found."
    exit 1
fi

make clean
find -name \*.c.smatch -exec rm \{\} \;
make -j${NR_CPU} $ENDIAN -k CHECK="$CMD -p=kernel --file-output --succeed $*" \
        C=1 $TARGET 2>&1 | tee $LOG
find -name \*.c.smatch -exec cat \{\} \; -exec rm \{\} \; > $WLOG
find -name \*.c.smatch.sql -exec cat \{\} \; -exec rm \{\} \; > $WLOG.sql
```



```
find -name \*.c.smatch.caller_info -exec cat \{\} \; -exec rm \{\} \; > $WLOG.ca

echo "Done.  The warnings are saved to $WLOG"
```

```
**********************************************************
    2078 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_scripts/trace_params.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl

# This script is supposed to help use the param_mapper output.
# Give it a function and parameter and it lists the functions
# and parameters which are basically equivalent.

use strict;

sub usage()
{
    print ("trace_params.pl <smatch output file> <function> <parameter>\n");
    exit(1);
}

my %param_map;

my $UNKNOWN  = 1;
my $NOTFOUND = 2;
my $FOUND    = 3;

sub recurse($$)
{
    my $link = shift;
    my $target = shift;
    my $found = 0;

    if ($link =~ /$target/) {
        $param_map{$link}->{found} = $FOUND;
        return 1;
    }

    if ($param_map{$link}->{found} == $FOUND) {
        return 1;
    }
    if ($param_map{$link}->{found} == $NOTFOUND) {
        return 0;
    }

    $param_map{$link}->{found} = $NOTFOUND;
    foreach my $l (@{$param_map{$link}->{links}}){
        $found = recurse($l, $target);
        if ($found) {
            $param_map{$link}->{found} = $FOUND;
            return 1;
        }
    }

    return 0;
}

sub compress_all($$)
{
    my $f = shift;
    my $p = shift;
    my $target = "$f%$p";

    foreach my $link (keys %param_map){
        recurse($link, $target);
    }
}
```



```
sub add_link($$)
{
    my $one = shift;
    my $two = shift;

    if (!defined($param_map{$one})) {
        $param_map{$one} = {found => $UNKNOWN, links => []};
    }
    push @{$param_map{$one}->{links}}, $two;
}

sub load_all($)
{
    my $file = shift;

    open(FILE, "<$file");
    while (<FILE>) {
        if (/.*?:\d+ (.*?)\(\) info: param_mapper (\d+) => (.*?) (\d+)/) {
            add_link("$1%$2", "$3%$4");
        }
    }
}

sub print_found()
{
    foreach my $func (keys %param_map){
        my $tmp = $param_map{$func};

        if ($tmp->{found} == $FOUND) {
            my ($f, $p) = split(/%/, $func);
            print("$f $p\n");
        }
    }
}

my $file = shift();
my $func = shift();
my $param = shift();

if (!$file or !$func or !defined($param)) {
    usage();
}

if (! -e $file) {
    printf("Error:  $file does not exist.\n");
    exit(1);
}

load_all($file);
compress_all($func, $param);
print_found();
```

```
**********************************************************
    1792 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_scripts/unlocked_paths.pl
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/usr/bin/perl

use strict;

sub usage()
{
    print "Usage: unlocked_paths.pl <call tree file> <lock> <function>\n";
    print "Prints a list of paths to <function> which don't take the <lock>.\n";
    print "Generate the call tree file by running smatch with --call-tree.\n";
    exit(1);
}

my %f_map;

sub add_to_map($)
{
    my $callee = shift;

    if (!defined($f_map{$callee})) {
        $f_map{$callee} = {visited => 0, called_by => {}};
    }
}

sub add_called_by($$)
{
    my $caller = shift;
    my $callee = shift;
    my $tmp;

    %{$f_map{$callee}->{called_by}}->{$caller} = 1;
}

sub load_all($$)
{
    my $file = shift;
    my $lock = shift;

    open(FILE, "<$file");
    while (<FILE>) {
        if (/.*?:\d+ (.*?)\(\) info: func_call \((.*)\) (.*)/) {
            my $caller = quotemeta $1;
            my $locks = quotemeta $2;
            my $callee = quotemeta $3;

            add_to_map($callee);
            if (!($locks =~ /$lock/)) {
                add_called_by($caller, $callee);
            }
        }
    }
}

my @fstack;
sub print_fstack()
{
    foreach my $f (reverse @fstack) {
        printf "$f ";
    }
    printf "\n";
}
```



```
sub print_unlocked_paths($)
{
    my $function = shift;

    if (! defined %{$f_map{$function}}->{called_by}) {
        push @fstack, $function;
        print_fstack();
        pop @fstack;
        return;
    }

    push @fstack, $function;

    if (!%{$f_map{$function}}->{visited}) {
        %{$f_map{$function}}->{visited} = 1;
        foreach my $caller (keys %{%{$f_map{$function}}->{called_by}}){
            print_unlocked_paths($caller);
        }
        %{$f_map{$function}}->{visited} = 0;

    }

    pop @fstack;
}

my $file = shift;
my $lock = shift;
my $target = shift;

if (!$file || !$lock || !$target) {
    usage();
}
if (! -e $file) {
    printf("Error:  $file does not exist.\n");
    exit(1);
}

load_all($file, $lock);
print_unlocked_paths($target);
```

```
**********************************************************
    1846 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_scripts/whitespace_only.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash -e

usage()
{
    echo "usage:  $0 <patch file>"
    exit 1
}

if [ "$1" = "" ] ; then
    usage
fi

if [ "$1" = "--compile" ] ; then
    compile=true
    shift
fi

SCRIPT_DIR=$(dirname $0)
if [ -e $SCRIPT_DIR/kchecker ] ; then
    KCHECKER=$SCRIPT_DIR/kchecker
    STRIP=$SCRIPT_DIR/strip_whitespace.pl
elif which kchecker | grep kchecker > /dev/null ; then
    KCHECKER=kchecker
    STRIP=strip_whitespace.pl
else
    echo "$SCRIPT_DIR"
    echo "kchecker script not found."
    exit 1
fi

PATCH=$1

files=$(grep ^+++ $PATCH | cut -f 1 | cut -b 5-)
if [ "$files" = "" ] ; then
    usage
fi

if ! cat $PATCH | patch -p1 --dry-run > /dev/null ; then
    echo "Couldn't apply patch"
    exit 1
fi

before=$(mktemp /tmp/before.XXXXXXXXXX)
after=$(mktemp /tmp/after.XXXXXXXXXX)
tmpfile=$(mktemp)

for file in $files ; do
    file=${file#*/}

    $STRIP $file > $before
    if [ "$compile" = "true" ] ; then
        if ! $KCHECKER --test-parsing --outfile=$before $file ; then
                echo "warning: compile failed."
        fi
        mv $before $tmpfile
        $STRIP $file > $before
        cat $tmpfile >> $before
    fi
    cat $PATCH | patch -p1
    $STRIP $file > $after
```



```
    if [ "$compile" = "true" ] ; then
        if ! $KCHECKER --test-parsing --outfile=$after $file ; then
                echo "warning: compile failed.  *again*"
        fi
        mv $after $tmpfile
        $STRIP $file > $after
        cat $tmpfile >> $after
    fi
    cat $PATCH | patch -p1 -R

    if [ ! -s $before ] ; then
        echo "Error:  No result"
        exit 1
    fi

    if diff $before $after > /dev/null ; then
        echo
        echo Only white space changed
        echo
    else
        echo '!!#$%@$%@^@#$^@#%@$%@$%@#%$@#%!!'
        echo '!!                              !!'
        echo '!!  This patch changes stuff  !!'
        echo '!!                              !!'
        echo '!!#$%@$%@^@#$^@#%@$%@$%@#%$@#%!!'

        diff -u $before $after
    fi
    rm -f $before $after $tmpfile
done
```

```
**********************************************************
    1112 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_scripts/wine_checker.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/bash

function usage {
    echo "Usage:  $0 [--sparse][--valgrind][--debug] path/to/file.c"
    exit 1
}

SCRIPT_DIR=$(dirname $0)
if [ -e $SCRIPT_DIR/../smatch ] ; then
    CMD=$SCRIPT_DIR/../smatch
elif which smatch | grep smatch > /dev/null ; then
    CMD=smatch
else
    echo "Smatch binary not found."
    exit 1
fi

POST=""
WINE_ARGS="-p=wine --full-path -D__i386__"

while true ; do
    if [[ "$1" == "--sparse" ]] ; then
        CMD="sparse"
        shift
    elif [[ "$1" == "--valgrind" ]] ; then
        PRE="valgrind"
        shift
    elif [[ "$1" == "" ]] ; then
        break
    else
        if [[ "$1" == "--help" ]] ; then
                $CMD --help
                exit 1
        fi
        if echo $1 | grep -q ^- ; then
                POST="$POST $1"
        else
                break
        fi
        shift
    fi
done

cname=$1
cname=$(echo ${cname/.o/.c})
if [[ "$cname" == "" ]] ; then
    usege
fi
if ! test -e $cname ; then
    usege
fi

oname=$(echo ${cname/.c/.o})
if ! echo $oname | grep .o$ > /dev/null ; then
    usege
fi
rm -f $oname

cur=$(pwd)
file_dir=$(dirname $oname)
```



```
o_short_name=$(basename $oname)
cd $file_dir
make CC="$PRE $CMD $POST $WINE_ARGS" $o_short_name
make $o_short_name
cd $cur
```

```
**********************************************************
   23512 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_slist.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2008,2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include <stdio.h>
#include "smatch.h"
#include "smatch_slist.h"

#undef CHECKORDER

ALLOCATOR(smatch_state, "smatch state");
ALLOCATOR(sm_state, "sm state");
ALLOCATOR(named_stree, "named slist");
__DO_ALLOCATOR(char, 1, 4, "state names", sname);

int sm_state_counter;

static struct stree_stack *all_pools;

const char *show_sm(struct sm_state *sm)
{
        static char buf[256];
        struct sm_state *tmp;
        int pos;
        int i;

        if (!sm)
                return "<none>";

        pos = snprintf(buf, sizeof(buf), "[%s] '%s' = '%s'",
                       check_name(sm->owner), sm->name, show_state(sm->state));
        if (pos > sizeof(buf))
                goto truncate;

        if (ptr_list_size((struct ptr_list *)sm->possible) == 1)
                return buf;

        pos += snprintf(buf + pos, sizeof(buf) - pos, " (");
        if (pos > sizeof(buf))
                goto truncate;
        i = 0;
        FOR_EACH_PTR(sm->possible, tmp) {
                if (i++)
                        pos += snprintf(buf + pos, sizeof(buf) - pos, ", ");
                if (pos > sizeof(buf))
                        goto truncate;
```



```
                pos += snprintf(buf + pos, sizeof(buf) - pos, "%s",
                                show_state(tmp->state));
                if (pos > sizeof(buf))
                        goto truncate;
        } END_FOR_EACH_PTR(tmp);
        snprintf(buf + pos, sizeof(buf) - pos, ")");

        return buf;

truncate:
        for (i = 0; i < 3; i++)
                buf[sizeof(buf) - 2 - i] = '.';
        return buf;
}

void __print_stree(struct stree *stree)
{
        struct sm_state *sm;

        printf("dumping stree at %d [%ld states]\n", get_lineno(), stree_count(s
        FOR_EACH_SM(stree, sm) {
                printf("%s\n", show_sm(sm));
        } END_FOR_EACH_SM(sm);
        printf("---\n");
}

/* NULL states go at the end to simplify merge_slist */
int cmp_tracker(const struct sm_state *a, const struct sm_state *b)
{
        int ret;

        if (a == b)
                return 0;
        if (!b)
                return -1;
        if (!a)
                return 1;

        if (a->owner > b->owner)
                return -1;
        if (a->owner < b->owner)
                return 1;

        ret = strcmp(a->name, b->name);
        if (ret < 0)
                return -1;
        if (ret > 0)
                return 1;

        if (!b->sym && a->sym)
                return -1;
        if (!a->sym && b->sym)
                return 1;
        if (a->sym < b->sym)
                return -1;
        if (a->sym > b->sym)
                return 1;

        return 0;
}

static int cmp_sm_states(const struct sm_state *a, const struct sm_state *b, int
{
        int ret;

        ret = cmp_tracker(a, b);
```

```
127         if (ret)
128                 return ret;

130         /* todo:  add hook for smatch_extra.c */
131         if (a->state > b->state)
132                 return -1;
133         if (a->state < b->state)
134                 return 1;
135         /* This is obviously a massive disgusting hack but we need to preserve
136          * the unmerged states for smatch extra because we use them in
137          * smatch_db.c.  Meanwhile if we preserve all the other unmerged states
138          * then it uses a lot of memory and we don't use it.  Hence this hack.
139          *
140          * Also sometimes even just preserving every possible SMATCH_EXTRA state
141          * takes too much resources so we have to cap that.  Capping is probably
142          * not often a problem in real life.
143          */
144         if (a->owner == SMATCH_EXTRA && preserve) {
145                 if (a == b)
146                         return 0;
147                 if (a->merged == 1 && b->merged == 0)
148                         return -1;
149                 if (a->merged == 0)
150                         return 1;
151         }

153         return 0;
154 }

156 struct sm_state *alloc_sm_state(int owner, const char *name,
157                                 struct symbol *sym, struct smatch_state *state)
158 {
159         struct sm_state *sm_state = __alloc_sm_state(0);

161         sm_state_counter++;

163         sm_state->name = alloc_sname(name);
164         sm_state->owner = owner;
165         sm_state->sym = sym;
166         sm_state->state = state;
167         sm_state->line = get_lineno();
168         sm_state->merged = 0;
169         sm_state->pool = NULL;
170         sm_state->left = NULL;
171         sm_state->right = NULL;
172         sm_state->nr_children = 1;
173         sm_state->possible = NULL;
174         add_ptr_list(&sm_state->possible, sm_state);
175         return sm_state;
176 }

178 static struct sm_state *alloc_state_no_name(int owner, const char *name,
179                                             struct symbol *sym,
180                                             struct smatch_state *state)
181 {
182         struct sm_state *tmp;

184         tmp = alloc_sm_state(owner, NULL, sym, state);
185         tmp->name = name;
186         return tmp;
187 }

189 int too_many_possible(struct sm_state *sm)
190 {
191         if (ptr_list_size((struct ptr_list *)sm->possible) >= 100)
192                 return 1;
```



```
193         return 0;
194 }

196 void add_possible_sm(struct sm_state *to, struct sm_state *new)
197 {
198         struct sm_state *tmp;
199         int preserve = 1;

201         if (too_many_possible(to))
202                 preserve = 0;

204         FOR_EACH_PTR(to->possible, tmp) {
205                 if (cmp_sm_states(tmp, new, preserve) < 0)
206                         continue;
207                 else if (cmp_sm_states(tmp, new, preserve) == 0) {
208                         return;
209                 } else {
210                         INSERT_CURRENT(new, tmp);
211                         return;
212                 }
213         } END_FOR_EACH_PTR(tmp);
214         add_ptr_list(&to->possible, new);
215 }

217 static void copy_possibles(struct sm_state *to, struct sm_state *from)
218 {
219         struct sm_state *tmp;

221         FOR_EACH_PTR(from->possible, tmp) {
222                 add_possible_sm(to, tmp);
223         } END_FOR_EACH_PTR(tmp);
224 }

226 char *alloc_sname(const char *str)
227 {
228         char *tmp;

230         if (!str)
231                 return NULL;
232         tmp = __alloc_sname(strlen(str) + 1);
233         strcpy(tmp, str);
234         return tmp;
235 }

237 int out_of_memory(void)
238 {
239         /*
240          * I decided to use 50M here based on trial and error.
241          * It works out OK for the kernel and so it should work
242          * for most other projects as well.
243          */
244         if (sm_state_counter * sizeof(struct sm_state) >= 100000000)
245                 return 1;
246         return 0;
247 }

249 int low_on_memory(void)
250 {
251         if (sm_state_counter * sizeof(struct sm_state) >= 25000000)
252                 return 1;
253         return 0;
254 }

256 static void free_sm_state(struct sm_state *sm)
257 {
258         free_slist(&sm->possible);
```

```
259         /*
260          * fixme.  Free the actual state.
261          * Right now we leave it until the end of the function
262          * because we don't want to double free it.
263          * Use the freelist to not double free things
264          */
265 }

267 static void free_all_sm_states(struct allocation_blob *blob)
268 {
269         unsigned int size = sizeof(struct sm_state);
270         unsigned int offset = 0;

272         while (offset < blob->offset) {
273                 free_sm_state((struct sm_state *)(blob->data + offset));
274                 offset += size;
275         }
276 }

278 /* At the end of every function we free all the sm_states */
279 void free_every_single_sm_state(void)
280 {
281         struct allocator_struct *desc = &sm_state_allocator;
282         struct allocation_blob *blob = desc->blobs;

284         desc->blobs = NULL;
285         desc->allocations = 0;
286         desc->total_bytes = 0;
287         desc->useful_bytes = 0;
288         desc->freelist = NULL;
289         while (blob) {
290                 struct allocation_blob *next = blob->next;
291                 free_all_sm_states(blob);
292                 blob_free(blob, desc->chunking);
293                 blob = next;
294         }
295         clear_sname_alloc();
296         clear_smatch_state_alloc();

298         free_stack_and_strees(&all_pools);
299         sm_state_counter = 0;
300 }

302 unsigned long get_pool_count(void)
303 {
304         return ptr_list_size((struct ptr_list *)all_pools);
305 }

307 struct sm_state *clone_sm(struct sm_state *s)
308 {
309         struct sm_state *ret;

311         ret = alloc_state_no_name(s->owner, s->name, s->sym, s->state);
312         ret->merged = s->merged;
313         ret->line = s->line;
314         /* clone_sm() doesn't copy the pools.  Each state needs to have
315            only one pool. */
316         ret->possible = clone_slist(s->possible);
317         ret->left = s->left;
318         ret->right = s->right;
319         ret->nr_children = s->nr_children;
320         return ret;
321 }

323 int is_merged(struct sm_state *sm)
324 {
```



```
325         return sm->merged;
326 }

328 int is_leaf(struct sm_state *sm)
329 {
330         return !sm->merged;
331 }

333 int slist_has_state(struct state_list *slist, struct smatch_state *state)
334 {
335         struct sm_state *tmp;

337         FOR_EACH_PTR(slist, tmp) {
338                 if (tmp->state == state)
339                         return 1;
340         } END_FOR_EACH_PTR(tmp);
341         return 0;
342 }

344 struct state_list *clone_slist(struct state_list *from_slist)
345 {
346         struct sm_state *sm;
347         struct state_list *to_slist = NULL;

349         FOR_EACH_PTR(from_slist, sm) {
350                 add_ptr_list(&to_slist, sm);
351         } END_FOR_EACH_PTR(sm);
352         return to_slist;
353 }

355 static struct smatch_state *merge_states(int owner, const char *name,
356                                          struct symbol *sym,
357                                          struct smatch_state *state1,
358                                          struct smatch_state *state2)
359 {
360         struct smatch_state *ret;

362         if (state1 == state2)
363                 ret = state1;
364         else if (__has_merge_function(owner))
365                 ret = __client_merge_function(owner, state1, state2);
366         else if (state1 == &ghost)
367                 ret = state2;
368         else if (state2 == &ghost)
369                 ret = state1;
370         else if (!state1 || !state2)
371                 ret = &undefined;
372         else
373                 ret = &merged;
374         return ret;
375 }

377 struct sm_state *merge_sm_states(struct sm_state *one, struct sm_state *two)
378 {
379         struct smatch_state *s;
380         struct sm_state *result;
381         static int warned;

383         if (one == two)
384                 return one;
385         if (out_of_memory()) {
386                 if (!warned)
387                         sm_warning("Function too hairy.  No more merges.");
388                 warned = 1;
389                 return one;
390         }
```

```
391         warned = 0;
392         s = merge_states(one->owner, one->name, one->sym, one->state, two->state
393         result = alloc_state_no_name(one->owner, one->name, one->sym, s);
394         result->merged = 1;
395         result->left = one;
396         result->right = two;
397         result->nr_children = one->nr_children + two->nr_children;
398         copy_possibles(result, one);
399         copy_possibles(result, two);

401         /*
402          * The ->line information is used by deref_check where we complain about
403          * checking pointers that have already been dereferenced.  Let's say we
404          * dereference a pointer on both the true and false paths and then merge
405          * the states here.  The result state is &derefed, but the ->line number
406          * is on the line where the pointer is merged not where it was
407          * dereferenced..
408          *
409          * So in that case, let's just pick one dereference and set the ->line
410          * to point at it.
411          *
412          */

414         if (result->state == one->state)
415                 result->line = one->line;
416         if (result->state == two->state)
417                 result->line = two->line;

419         if (option_debug ||
420             strcmp(check_name(one->owner), option_debug_check) == 0) {
421                 struct sm_state *tmp;
422                 int i = 0;

424                 printf("%s:%d %s() merge [%s] '%s' %s(L %d) + %s(L %d) => %s (",
425                         get_filename(), get_lineno(), get_function(),
426                         check_name(one->owner), one->name,
427                         show_state(one->state), one->line,
428                         show_state(two->state), two->line,
429                         show_state(s));

431                 FOR_EACH_PTR(result->possible, tmp) {
432                         if (i++)
433                                 printf(", ");
434                         printf("%s", show_state(tmp->state));
435                 } END_FOR_EACH_PTR(tmp);
436                 printf(")\n");
437         }

439         return result;
440 }

442 struct sm_state *get_sm_state_stree(struct stree *stree, int owner, const char *
443                                     struct symbol *sym)
444 {
445         struct tracker tracker = {
446                 .owner = owner,
447                 .name = (char *)name,
448                 .sym = sym,
449         };

451         if (!name)
452                 return NULL;


455         return avl_lookup(stree, (struct sm_state *)&tracker);
456 }
```



```
458 struct smatch_state *get_state_stree(struct stree *stree,
459                                     int owner, const char *name,
460                                     struct symbol *sym)
461 {
462         struct sm_state *sm;

464         sm = get_sm_state_stree(stree, owner, name, sym);
465         if (sm)
466                 return sm->state;
467         return NULL;
468 }

470 /* FIXME: this is almost exactly the same as set_sm_state_slist() */
471 void overwrite_sm_state_stree(struct stree **stree, struct sm_state *new)
472 {
473         avl_insert(stree, new);
474 }

476 void overwrite_sm_state_stree_stack(struct stree_stack **stack,
477                                     struct sm_state *sm)
478 {
479         struct stree *stree;

481         stree = pop_stree(stack);
482         overwrite_sm_state_stree(&stree, sm);
483         push_stree(stack, stree);
484 }

486 struct sm_state *set_state_stree(struct stree **stree, int owner, const char *na
487                                  struct symbol *sym, struct smatch_state *state)
488 {
489         struct sm_state *new = alloc_sm_state(owner, name, sym, state);

491         avl_insert(stree, new);
492         return new;
493 }

495 void set_state_stree_perm(struct stree **stree, int owner, const char *name,
496                                  struct symbol *sym, struct smatch_state *state)
497 {
498         struct sm_state *sm;

500         sm = malloc(sizeof(*sm) + strlen(name) + 1);
501         memset(sm, 0, sizeof(*sm));
502         sm->owner = owner;
503         sm->name = (char *)(sm + 1);
504         strcpy((char *)sm->name, name);
505         sm->sym = sym;
506         sm->state = state;

508         overwrite_sm_state_stree(stree, sm);
509 }

511 void delete_state_stree(struct stree **stree, int owner, const char *name,
512                                  struct symbol *sym)
513 {
514         struct tracker tracker = {
515                 .owner = owner,
516                 .name = (char *)name,
517                 .sym = sym,
518         };

520         avl_remove(stree, (struct sm_state *)&tracker);
521 }
```

```
523 void delete_state_stree_stack(struct stree_stack **stack, int owner, const char
524                         struct symbol *sym)
525 {
526         struct stree *stree;
527
528         stree = pop_stree(stack);
529         delete_state_stree(&stree, owner, name, sym);
530         push_stree(stack, stree);
531 }
532
533 void push_stree(struct stree_stack **stack, struct stree *stree)
534 {
535         add_ptr_list(stack, stree);
536 }
537
538 struct stree *pop_stree(struct stree_stack **stack)
539 {
540         struct stree *stree;
541
542         stree = last_ptr_list((struct ptr_list *)*stack);
543         delete_ptr_list_last((struct ptr_list **)stack);
544         return stree;
545 }
546
547 struct stree *top_stree(struct stree_stack *stack)
548 {
549         return last_ptr_list((struct ptr_list *)stack);
550 }
551
552 void free_slist(struct state_list **slist)
553 {
554         __free_ptr_list((struct ptr_list **)slist);
555 }
556
557 void free_stree_stack(struct stree_stack **stack)
558 {
559         __free_ptr_list((struct ptr_list **)stack);
560 }
561
562 void free_stack_and_strees(struct stree_stack **stree_stack)
563 {
564         struct stree *stree;
565
566         FOR_EACH_PTR(*stree_stack, stree) {
567                 free_stree(&stree);
568         } END_FOR_EACH_PTR(stree);
569         free_stree_stack(stree_stack);
570 }
571
572 struct sm_state *set_state_stree_stack(struct stree_stack **stack, int owner, co
573                                 struct symbol *sym, struct smatch_state *state)
574 {
575         struct stree *stree;
576         struct sm_state *sm;
577
578         stree = pop_stree(stack);
579         sm = set_state_stree(&stree, owner, name, sym, state);
580         push_stree(stack, stree);
581
582         return sm;
583 }
584
585 /*
586  * get_sm_state_stack() gets the state for the top slist on the stack.
587  */
588 struct sm_state *get_sm_state_stree_stack(struct stree_stack *stack,
```



```
589                                 int owner, const char *name,
590                                 struct symbol *sym)
591 {
592         struct stree *stree;
593         struct sm_state *ret;
594
595         stree = pop_stree(&stack);
596         ret = get_sm_state_stree(stree, owner, name, sym);
597         push_stree(&stack, stree);
598         return ret;
599 }
600
601 struct smatch_state *get_state_stree_stack(struct stree_stack *stack,
602                                 int owner, const char *name,
603                                 struct symbol *sym)
604 {
605         struct sm_state *sm;
606
607         sm = get_sm_state_stree_stack(stack, owner, name, sym);
608         if (sm)
609                 return sm->state;
610         return NULL;
611 }
612
613 static void match_states_stree(struct stree **one, struct stree **two)
614 {
615         struct smatch_state *tmp_state;
616         struct sm_state *sm;
617         struct state_list *add_to_one = NULL;
618         struct state_list *add_to_two = NULL;
619         AvlIter one_iter;
620         AvlIter two_iter;
621
622         avl_iter_begin(&one_iter, *one, FORWARD);
623         avl_iter_begin(&two_iter, *two, FORWARD);
624
625         for (;;) {
626                 if (!one_iter.sm && !two_iter.sm)
627                         break;
628                 if (cmp_tracker(one_iter.sm, two_iter.sm) < 0) {
629                         __set_fake_cur_stree_fast(*two);
630                         tmp_state = __client_unmatched_state_function(one_iter.s
631                         __pop_fake_cur_stree_fast();
632                         sm = alloc_state_no_name(one_iter.sm->owner, one_iter.sm
633                                                 one_iter.sm->sym, tmp_state);
634                         add_ptr_list(&add_to_two, sm);
635                         avl_iter_next(&one_iter);
636                 } else if (cmp_tracker(one_iter.sm, two_iter.sm) == 0) {
637                         avl_iter_next(&one_iter);
638                         avl_iter_next(&two_iter);
639                 } else {
640                         __set_fake_cur_stree_fast(*one);
641                         tmp_state = __client_unmatched_state_function(two_iter.s
642                         __pop_fake_cur_stree_fast();
643                         sm = alloc_state_no_name(two_iter.sm->owner, two_iter.sm
644                                                 two_iter.sm->sym, tmp_state);
645                         add_ptr_list(&add_to_one, sm);
646                         avl_iter_next(&two_iter);
647                 }
648         }
649
650         FOR_EACH_PTR(add_to_one, sm) {
651                 avl_insert(one, sm);
652         } END_FOR_EACH_PTR(sm);
653
654         FOR_EACH_PTR(add_to_two, sm) {
```

```
655                 avl_insert(two, sm);
656         } END_FOR_EACH_PTR(sm);

658         free_slist(&add_to_one);
659         free_slist(&add_to_two);
660 }

662 static void call_pre_merge_hooks(struct stree **one, struct stree **two)
663 {
664         struct sm_state *sm, *other;

666         save_all_states();

668         __swap_cur_stree(*one);
669         FOR_EACH_SM(*two, sm) {
670                 other = get_sm_state(sm->owner, sm->name, sm->sym);
671                 if (other == sm)
672                         continue;
673                 call_pre_merge_hook(sm);
674         } END_FOR_EACH_SM(sm);
675         *one = clone_stree(__get_cur_stree());

677         __swap_cur_stree(*two);
678         FOR_EACH_SM(*one, sm) {
679                 other = get_sm_state(sm->owner, sm->name, sm->sym);
680                 if (other == sm)
681                         continue;
682                 call_pre_merge_hook(sm);
683         } END_FOR_EACH_SM(sm);
684         *two = clone_stree(__get_cur_stree());

686         restore_all_states();
687 }

689 static void clone_pool_havers_stree(struct stree **stree)
690 {
691         struct sm_state *sm, *tmp;
692         struct state_list *slist = NULL;

694         FOR_EACH_SM(*stree, sm) {
695                 if (sm->pool) {
696                         tmp = clone_sm(sm);
697                         add_ptr_list(&slist, tmp);
698                 }
699         } END_FOR_EACH_SM(sm);

701         FOR_EACH_PTR(slist, sm) {
702                 avl_insert(stree, sm);
703         } END_FOR_EACH_PTR(sm);

705         free_slist(&slist);
706 }

708 int __stree_id;

710 /*
711  * merge_slist() is called whenever paths merge, such as after
712  * an if statement.  It takes the two slists and creates one.
713  */
714 static void __merge_stree(struct stree **to, struct stree *stree, int add_pool)
715 {
716         struct stree *results = NULL;
717         struct stree *implied_one = NULL;
718         struct stree *implied_two = NULL;
719         AvlIter one_iter;
720         AvlIter two_iter;
```



```
721         struct sm_state *tmp_sm;

723         if (out_of_memory())
724                 return;

726         /* merging a null and nonnull path gives you only the nonnull path */
727         if (!stree)
728                 return;
729         if (*to == stree)
730                 return;

732         if (!*to) {
733                 *to = clone_stree(stree);
734                 return;
735         }

737         implied_one = clone_stree(*to);
738         implied_two = clone_stree(stree);

740         match_states_stree(&implied_one, &implied_two);
741         call_pre_merge_hooks(&implied_one, &implied_two);

743         if (add_pool) {
744                 clone_pool_havers_stree(&implied_one);
745                 clone_pool_havers_stree(&implied_two);

747                 set_stree_id(&implied_one, ++__stree_id);
748                 set_stree_id(&implied_two, ++__stree_id);
749                 if (implied_one->base_stree)
750                         set_stree_id(&implied_one->base_stree, ++__stree_id);
751                 if (implied_two->base_stree)
752                         set_stree_id(&implied_two->base_stree, ++__stree_id);
753         }

755         push_stree(&all_pools, implied_one);
756         push_stree(&all_pools, implied_two);

758         avl_iter_begin(&one_iter, implied_one, FORWARD);
759         avl_iter_begin(&two_iter, implied_two, FORWARD);

761         for (;;) {
762                 if (!one_iter.sm || !two_iter.sm)
763                         break;
764                 if (cmp_tracker(one_iter.sm, two_iter.sm) < 0) {
765                         sm_perror(" in %s", __func__);
766                         avl_iter_next(&one_iter);
767                 } else if (cmp_tracker(one_iter.sm, two_iter.sm) == 0) {
768                         if (add_pool && one_iter.sm != two_iter.sm) {
769                                 one_iter.sm->pool = implied_one;
770                                 if (implied_one->base_stree)
771                                         one_iter.sm->pool = implied_one->base_st
772                                 two_iter.sm->pool = implied_two;
773                                 if (implied_two->base_stree)
774                                         two_iter.sm->pool = implied_two->base_st
775                         }
776                         tmp_sm = merge_sm_states(one_iter.sm, two_iter.sm);
777                         add_possible_sm(tmp_sm, one_iter.sm);
778                         add_possible_sm(tmp_sm, two_iter.sm);
779                         avl_insert(&results, tmp_sm);
780                         avl_iter_next(&one_iter);
781                         avl_iter_next(&two_iter);
782                 } else {
783                         sm_perror(" in %s", __func__);
784                         avl_iter_next(&two_iter);
785                 }
786         }
```

```
788         free_stree(to);
789         *to = results;
790 }

792 void merge_stree(struct stree **to, struct stree *stree)
793 {
794         __merge_stree(to, stree, 1);
795 }

797 void merge_stree_no_pools(struct stree **to, struct stree *stree)
798 {
799         __merge_stree(to, stree, 0);
800 }

802 /*
803  * This is unfortunately a bit subtle...  The problem is that if a
804  * state is set on one fake stree but not the other then we should
805  * look up the the original state and use that as the unset state.
806  * Fortunately, after you pop your fake stree then the cur_slist should
807  * reflect the original state.
808  */
809 void merge_fake_stree(struct stree **to, struct stree *stree)
810 {
811         struct stree *one = *to;
812         struct stree *two = stree;
813         struct sm_state *sm;
814         struct state_list *add_to_one = NULL;
815         struct state_list *add_to_two = NULL;
816         AvlIter one_iter;
817         AvlIter two_iter;

819         if (!stree)
820                 return;
821         if (*to == stree)
822                 return;
823         if (!*to) {
824                 *to = clone_stree(stree);
825                 return;
826         }

828         avl_iter_begin(&one_iter, one, FORWARD);
829         avl_iter_begin(&two_iter, two, FORWARD);

831         for (;;) {
832                 if (!one_iter.sm && !two_iter.sm)
833                         break;
834                 if (cmp_tracker(one_iter.sm, two_iter.sm) < 0) {
835                         sm = get_sm_state(one_iter.sm->owner, one_iter.sm->name,
836                                           one_iter.sm->sym);
837                         if (sm)
838                                 add_ptr_list(&add_to_two, sm);
839                         avl_iter_next(&one_iter);
840                 } else if (cmp_tracker(one_iter.sm, two_iter.sm) == 0) {
841                         avl_iter_next(&one_iter);
842                         avl_iter_next(&two_iter);
843                 } else {
844                         sm = get_sm_state(two_iter.sm->owner, two_iter.sm->name,
845                                           two_iter.sm->sym);
846                         if (sm)
847                                 add_ptr_list(&add_to_one, sm);
848                         avl_iter_next(&two_iter);
849                 }
850         }

852         FOR_EACH_PTR(add_to_one, sm) {
```



```
853                 avl_insert(&one, sm);
854         } END_FOR_EACH_PTR(sm);

856         FOR_EACH_PTR(add_to_two, sm) {
857                 avl_insert(&two, sm);
858         } END_FOR_EACH_PTR(sm);

860         one->base_stree = clone_stree(__get_cur_stree());
861         FOR_EACH_SM(one, sm) {
862                 avl_insert(&one->base_stree, sm);
863         } END_FOR_EACH_SM(sm);

865         two->base_stree = clone_stree(__get_cur_stree());
866         FOR_EACH_SM(two, sm) {
867                 avl_insert(&two->base_stree, sm);
868         } END_FOR_EACH_SM(sm);

870         free_slist(&add_to_one);
871         free_slist(&add_to_two);

873         __merge_stree(&one, two, 1);

875         *to = one;
876 }

878 /*
879  * filter_slist() removes any sm states "slist" holds in common with "filter"
880  */
881 void filter_stree(struct stree **stree, struct stree *filter)
882 {
883         struct stree *results = NULL;
884         AvlIter one_iter;
885         AvlIter two_iter;

887         avl_iter_begin(&one_iter, *stree, FORWARD);
888         avl_iter_begin(&two_iter, filter, FORWARD);

890         /* FIXME: This should probably be re-written with trees in mind */

892         for (;;) {
893                 if (!one_iter.sm && !two_iter.sm)
894                         break;
895                 if (cmp_tracker(one_iter.sm, two_iter.sm) < 0) {
896                         avl_insert(&results, one_iter.sm);
897                         avl_iter_next(&one_iter);
898                 } else if (cmp_tracker(one_iter.sm, two_iter.sm) == 0) {
899                         if (one_iter.sm != two_iter.sm)
900                                 avl_insert(&results, one_iter.sm);
901                         avl_iter_next(&one_iter);
902                         avl_iter_next(&two_iter);
903                 } else {
904                         avl_iter_next(&two_iter);
905                 }
906         }

908         free_stree(stree);
909         *stree = results;
910 }


913 /*
914  * and_slist_stack() pops the top two slists, overwriting the one with
915  * the other and pushing it back on the stack.
916  */
917 void and_stree_stack(struct stree_stack **stack)
918 {
```

```
919         struct sm_state *tmp;
920         struct stree *right_stree = pop_stree(stack);

922         FOR_EACH_SM(right_stree, tmp) {
923                 overwrite_sm_state_stree_stack(stack, tmp);
924         } END_FOR_EACH_SM(tmp);
925         free_stree(&right_stree);
926 }

928 /*
929  * or_slist_stack() is for if we have:  if (foo || bar) { foo->baz;
930  * It pops the two slists from the top of the stack and merges them
931  * together in a way that preserves the things they have in common
932  * but creates a merged state for most of the rest.
933  * You could have code that had:  if (foo || foo) { foo->baz;
934  * It's this function which ensures smatch does the right thing.
935  */
936 void or_stree_stack(struct stree_stack **pre_conds,
937                     struct stree *cur_stree,
938                     struct stree_stack **stack)
939 {
940         struct stree *new;
941         struct stree *old;
942         struct stree *pre_stree;
943         struct stree *res;
944         struct stree *tmp_stree;

946         new = pop_stree(stack);
947         old = pop_stree(stack);

949         pre_stree = pop_stree(pre_conds);
950         push_stree(pre_conds, clone_stree(pre_stree));

952         res = clone_stree(pre_stree);
953         overwrite_stree(old, &res);

955         tmp_stree = clone_stree(cur_stree);
956         overwrite_stree(new, &tmp_stree);

958         merge_stree(&res, tmp_stree);
959         filter_stree(&res, pre_stree);

961         push_stree(stack, res);
962         free_stree(&tmp_stree);
963         free_stree(&pre_stree);
964         free_stree(&new);
965         free_stree(&old);
966 }

968 /*
969  * get_named_stree() is only used for gotos.
970  */
971 struct stree **get_named_stree(struct named_stree_stack *stack,
972                                const char *name,
973                                struct symbol *sym)
974 {
975         struct named_stree *tmp;

977         FOR_EACH_PTR(stack, tmp) {
978                 if (tmp->sym == sym &&
979                     strcmp(tmp->name, name) == 0)
980                         return &tmp->stree;
981         } END_FOR_EACH_PTR(tmp);
982         return NULL;
983 }
```



```
985 /* FIXME:  These parameters are in a different order from expected */
986 void overwrite_stree(struct stree *from, struct stree **to)
987 {
988         struct sm_state *tmp;

990         FOR_EACH_SM(from, tmp) {
991                 overwrite_sm_state_stree(to, tmp);
992         } END_FOR_EACH_SM(tmp);
993 }
```

```
**********************************************************
    3706 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_slist.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct stree;

extern int unfree_stree;

DECLARE_PTR_LIST(state_list, struct sm_state);
DECLARE_PTR_LIST(state_list_stack, struct state_list);

struct named_stree {
        char *name;
        struct symbol *sym;
        struct stree *stree;
};
DECLARE_ALLOCATOR(named_stree);
DECLARE_PTR_LIST(named_stree_stack, struct named_stree);


extern struct state_list_stack *implied_pools;
extern int __stree_id;
extern int sm_state_counter;

const char *show_sm(struct sm_state *sm);
void __print_stree(struct stree *stree);
void add_history(struct sm_state *sm);
int cmp_tracker(const struct sm_state *a, const struct sm_state *b);
char *alloc_sname(const char *str);
struct sm_state *alloc_sm_state(int owner, const char *name,
                                struct symbol *sym, struct smatch_state *state);

void free_every_single_sm_state(void);
struct sm_state *clone_sm(struct sm_state *s);
int is_merged(struct sm_state *sm);
int is_leaf(struct sm_state *sm);
struct state_list *clone_slist(struct state_list *from_slist);

int slist_has_state(struct state_list *slist, struct smatch_state *state);

int too_many_possible(struct sm_state *sm);
void add_possible_sm(struct sm_state *to, struct sm_state *new);
struct sm_state *merge_sm_states(struct sm_state *one, struct sm_state *two);
struct smatch_state *get_state_stree(struct stree *stree, int owner, const char
                    struct symbol *sym);

struct sm_state *get_sm_state_stree(struct stree *stree, int owner, const char *
                    struct symbol *sym);

void overwrite_sm_state_stree(struct stree **stree, struct sm_state *sm);
void overwrite_sm_state_stree_stack(struct stree_stack **stack, struct sm_state
struct sm_state *set_state_stree(struct stree **stree, int owner, const char *na
                     struct symbol *sym, struct smatch_state *state);
void set_state_stree_perm(struct stree **stree, int owner, const char *name,
                     struct symbol *sym, struct smatch_state *state);
void delete_state_stree(struct stree **stree, int owner, const char *name,
                        struct symbol *sym);

void delete_state_stree_stack(struct stree_stack **stack, int owner, const char
                        struct symbol *sym);

void push_stree(struct stree_stack **list_stack, struct stree *stree);
struct stree *pop_stree(struct stree_stack **list_stack);
struct stree *top_stree(struct stree_stack *stack);
```



```
void free_slist(struct state_list **slist);
void free_stree_stack(struct stree_stack **stack);
void free_stack_and_strees(struct stree_stack **stree_stack);
unsigned long get_pool_count(void);

struct sm_state *set_state_stree_stack(struct stree_stack **stack, int owner, co
                                struct symbol *sym, struct smatch_state *state);

struct sm_state *get_sm_state_stree_stack(struct stree_stack *stack,
                                int owner, const char *name,
                                struct symbol *sym);
struct smatch_state *get_state_stree_stack(struct stree_stack *stack, int owner,
                                const char *name, struct symbol *sym);

int out_of_memory(void);
int low_on_memory(void);
void merge_stree(struct stree **to, struct stree *stree);
void merge_stree_no_pools(struct stree **to, struct stree *stree);
void merge_stree(struct stree **to, struct stree *right);
void merge_fake_stree(struct stree **to, struct stree *stree);
void filter_stree(struct stree **stree, struct stree *filter);
void and_stree_stack(struct stree_stack **stree_stack);

void or_stree_stack(struct stree_stack **pre_conds,
                    struct stree *cur_stree,
                    struct stree_stack **stack);

struct stree **get_named_stree(struct named_stree_stack *stack,
                               const char *name,
                               struct symbol *sym);

void overwrite_stree(struct stree *from, struct stree **to);

/* add stuff smatch_returns.c here */

void all_return_states_hook(void (*callback)(void));
```

```
**********************************************************
    1688 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_start_states.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Store the states at the start of the function because this is something that
 * is used in a couple places.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;

static struct stree *start_states;
static struct stree_stack *saved_stack;
static void save_start_states(struct statement *stmt)
{
        start_states = clone_stree(__get_cur_stree());
}

static void match_save_states(struct expression *expr)
{
        push_stree(&saved_stack, start_states);
        start_states = NULL;
}

static void match_restore_states(struct expression *expr)
{
        free_stree(&start_states);
        start_states = pop_stree(&saved_stack);
}

static void match_end_func(void)
{
        free_stree(&start_states);
}

struct stree *get_start_states(void)
{
        return start_states;
}

void register_start_states(int id)
{
        my_id = id;
```



```
        add_hook(&save_start_states, AFTER_DEF_HOOK);
        add_hook(&match_save_states, INLINE_FN_START);
        add_hook(&match_restore_states, INLINE_FN_END);
        add_hook(&match_end_func, AFTER_FUNC_HOOK);
}
```

```
**********************************************************
    2105 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_statement_count.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2018 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static struct smatch_state *merge_states(struct smatch_state *s1, struct smatch_
{
        int left, right, min;

        left = PTR_INT(s1->data);
        right = PTR_INT(s2->data);

        min = left;
        if (right < min)
                min = right;
        return alloc_state_num(min);
}

long get_stmt_cnt(void)
{
        struct smatch_state *state;

        state = get_state(my_id, "stmts", NULL);
        if (!state)
                return 0;
        return (long)state->data;
}

static void match_statement(struct statement *stmt)
{
        int cnt;

        cnt = get_stmt_cnt();
        cnt++;
        set_state(my_id, "stmts", NULL, alloc_state_num(cnt));
}

static void insert_return_info(int return_id, char *return_ranges, struct expres
{
        char buf[32];
```



```
        int cnt;

        cnt = get_stmt_cnt();
        snprintf(buf, sizeof(buf), "%d", cnt);
        sql_insert_return_states(return_id, return_ranges, STMT_CNT, -1, "", buf
}

static void select_return_info(struct expression *expr, int param, char *key, ch
{
        int cnt, add;

        cnt = get_stmt_cnt();
        add = atoi(value);

        set_state(my_id, "stmts", NULL, alloc_state_num(cnt + add));
}

void register_statement_count(int id)
{
        my_id = id;

        add_hook(match_statement, STMT_HOOK);
        add_merge_hook(my_id, &merge_states);

        add_split_return_callback(&insert_return_info);
        select_return_states_hook(STMT_CNT, &select_return_info);
}
```

```
**********************************************************
   25962 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_states.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2006 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * You have a lists of states.  kernel = locked, foo = NULL, ...
 * When you hit an if {} else {} statement then you swap the list
 * of states for a different list of states.  The lists are stored
 * on stacks.
 *
 * At the beginning of this file there are list of the stacks that
 * we use.  Each function in this file does something to one of
 * of the stacks.
 *
 * So the smatch_flow.c understands code but it doesn't understand states.
 * smatch_flow calls functions in this file.  This file calls functions
 * in smatch_slist.c which just has boring generic plumbing for handling
 * state lists.  But really it's this file where all the magic happens.
 */

#include <stdlib.h>
#include <stdio.h>
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

struct smatch_state undefined = { .name = "undefined" };
struct smatch_state ghost = { .name = "ghost" };
struct smatch_state merged = { .name = "merged" };
struct smatch_state true_state = { .name = "true" };
struct smatch_state false_state = { .name = "false" };

static struct stree *cur_stree; /* current states */

static struct stree_stack *true_stack; /* states after a t/f branch */
static struct stree_stack *false_stack;
static struct stree_stack *pre_cond_stack; /* states before a t/f branch */

static struct stree_stack *cond_true_stack; /* states affected by a branch */
static struct stree_stack *cond_false_stack;

static struct stree_stack *fake_cur_stree_stack;
static int read_only;

static struct stree_stack *break_stack;
static struct stree_stack *fake_break_stack;
static struct stree_stack *switch_stack;
```



```
static struct range_list_stack *remaining_cases;
static struct stree_stack *default_stack;
static struct stree_stack *continue_stack;

static struct named_stree_stack *goto_stack;

static struct ptr_list *backup;

int option_debug;

void __print_cur_stree(void)
{
        __print_stree(cur_stree);
}

int unreachable(void)
{
        if (!cur_stree)
                return 1;
        return 0;
}

struct sm_state *set_state(int owner, const char *name, struct symbol *sym, stru
{
        struct sm_state *ret;

        if (!name)
                return NULL;

        if (read_only)
                sm_perror("cur_stree is read only.");

        if (option_debug || strcmp(check_name(owner), option_debug_check) == 0)
                struct smatch_state *s;

                s = get_state(owner, name, sym);
                if (!s)
                        sm_msg("%s new [%s] '%s' %s", __func__,
                               check_name(owner), name, show_state(state));
                else
                        sm_msg("%s change [%s] '%s' %s => %s",
                               __func__, check_name(owner), name, show_state(s)
                               show_state(state));
        }

        if (owner != -1 && unreachable())
                return NULL;

        if (fake_cur_stree_stack)
                set_state_stree_stack(&fake_cur_stree_stack, owner, name, sym, s

        ret = set_state_stree(&cur_stree, owner, name, sym, state);

        return ret;
}

struct sm_state *set_state_expr(int owner, struct expression *expr, struct smatc
{
        char *name;
        struct symbol *sym;
        struct sm_state *ret = NULL;

        expr = strip_expr(expr);
        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                goto free;
```

```
127         ret = set_state(owner, name, sym, state);
128 free:
129         free_string(name);
130         return ret;
131 }

133 void __swap_cur_stree(struct stree *stree)
134 {
135         free_stree(&cur_stree);
136         cur_stree = stree;
137 }

139 void __push_fake_cur_stree(void)
140 {
141         push_stree(&fake_cur_stree_stack, NULL);
142         __save_pre_cond_states();
143 }

145 struct stree *__pop_fake_cur_stree(void)
146 {
147         if (!fake_cur_stree_stack)
148                 sm_perror("popping too many fake cur strees.");
149         __use_pre_cond_states();
150         return pop_stree(&fake_cur_stree_stack);
151 }

153 void __free_fake_cur_stree(void)
154 {
155         struct stree *stree;

157         stree = __pop_fake_cur_stree();
158         free_stree(&stree);
159 }

161 void __set_fake_cur_stree_fast(struct stree *stree)
162 {
163         push_stree(&pre_cond_stack, cur_stree);
164         cur_stree = stree;
165         read_only = 1;
166 }

168 void __pop_fake_cur_stree_fast(void)
169 {
170         cur_stree = pop_stree(&pre_cond_stack);
171         read_only = 0;
172 }

174 void __merge_stree_into_cur(struct stree *stree)
175 {
176         struct sm_state *sm;
177         struct sm_state *orig;
178         struct sm_state *merged;

180         FOR_EACH_SM(stree, sm) {
181                 orig = get_sm_state(sm->owner, sm->name, sm->sym);
182                 if (orig)
183                         merged = merge_sm_states(orig, sm);
184                 else
185                         merged = sm;
186                 __set_sm(merged);
187         } END_FOR_EACH_SM(sm);
188 }

190 void __set_sm(struct sm_state *sm)
191 {
192         if (read_only)
```



```
193                 sm_perror("cur_stree is read only.");

195         if (option_debug ||
196             strcmp(check_name(sm->owner), option_debug_check) == 0) {
197                 struct smatch_state *s;

199                 s = get_state(sm->owner, sm->name, sm->sym);
200                 if (!s)
201                         sm_msg("%s new %s", __func__, show_sm(sm));
202                 else
203                         sm_msg("%s change %s (was %s)", __func__, show_sm(sm),
204                                show_state(s));
205         }

207         if (unreachable())
208                 return;

210         if (fake_cur_stree_stack)
211                 overwrite_sm_state_stree_stack(&fake_cur_stree_stack, sm);

213         overwrite_sm_state_stree(&cur_stree, sm);
214 }

216 void __set_sm_cur_stree(struct sm_state *sm)
217 {
218         if (read_only)
219                 sm_perror("cur_stree is read only.");

221         if (option_debug ||
222             strcmp(check_name(sm->owner), option_debug_check) == 0) {
223                 struct smatch_state *s;

225                 s = get_state(sm->owner, sm->name, sm->sym);
226                 if (!s)
227                         sm_msg("%s new %s", __func__, show_sm(sm));
228                 else
229                         sm_msg("%s change %s (was %s)",
230                                __func__, show_sm(sm), show_state(s));
231         }

233         if (unreachable())
234                 return;

236         overwrite_sm_state_stree(&cur_stree, sm);
237 }

239 void __set_sm_fake_stree(struct sm_state *sm)
240 {
241         if (read_only)
242                 sm_perror("cur_stree is read only.");

244         if (option_debug ||
245             strcmp(check_name(sm->owner), option_debug_check) == 0) {
246                 struct smatch_state *s;

248                 s = get_state(sm->owner, sm->name, sm->sym);
249                 if (!s)
250                         sm_msg("%s new %s", __func__, show_sm(sm));
251                 else
252                         sm_msg("%s change %s (was %s)",
253                                __func__, show_sm(sm), show_state(s));
254         }

256         if (unreachable())
257                 return;
```

```
259         overwrite_sm_state_stree_stack(&fake_cur_stree_stack, sm);
260 }


263 typedef void (get_state_hook)(int owner, const char *name, struct symbol *sym);
264 DECLARE_PTR_LIST(fn_list, get_state_hook *);
265 static struct fn_list *get_state_hooks;

267 void add_get_state_hook(get_state_hook *fn)
268 {
269         get_state_hook **p = malloc(sizeof(get_state_hook *));
270         *p = fn;
271         add_ptr_list(&get_state_hooks, p);
272 }

274 static void call_get_state_hooks(int owner, const char *name, struct symbol *sym
275 {
276         static int recursion;
277         get_state_hook **fn;

279         if (recursion)
280                 return;
281         recursion = 1;

283         FOR_EACH_PTR(get_state_hooks, fn) {
284                 (*fn)(owner, name, sym);
285         } END_FOR_EACH_PTR(fn);

287         recursion = 0;
288 }

290 struct smatch_state *__get_state(int owner, const char *name, struct symbol *sym
291 {
292         return get_state_stree(cur_stree, owner, name, sym);
293 }

295 struct smatch_state *get_state(int owner, const char *name, struct symbol *sym)
296 {
297         call_get_state_hooks(owner, name, sym);

299         return __get_state(owner, name, sym);
300 }

302 struct smatch_state *get_state_expr(int owner, struct expression *expr)
303 {
304         char *name;
305         struct symbol *sym;
306         struct smatch_state *ret = NULL;

308         expr = strip_expr(expr);
309         name = expr_to_var_sym(expr, &sym);
310         if (!name || !sym)
311                 goto free;
312         ret = get_state(owner, name, sym);
313 free:
314         free_string(name);
315         return ret;
316 }

318 struct state_list *get_possible_states(int owner, const char *name, struct symbo
319 {
320         struct sm_state *sms;

322         sms = get_sm_state_stree(cur_stree, owner, name, sym);
323         if (sms)
324                 return sms->possible;
```



```
325         return NULL;
326 }

328 struct state_list *get_possible_states_expr(int owner, struct expression *expr)
329 {
330         char *name;
331         struct symbol *sym;
332         struct state_list *ret = NULL;

334         expr = strip_expr(expr);
335         name = expr_to_var_sym(expr, &sym);
336         if (!name || !sym)
337                 goto free;
338         ret = get_possible_states(owner, name, sym);
339 free:
340         free_string(name);
341         return ret;
342 }

344 struct sm_state *get_sm_state(int owner, const char *name, struct symbol *sym)
345 {
346         return get_sm_state_stree(cur_stree, owner, name, sym);
347 }

349 struct sm_state *get_sm_state_expr(int owner, struct expression *expr)
350 {
351         char *name;
352         struct symbol *sym;
353         struct sm_state *ret = NULL;

355         expr = strip_expr(expr);
356         name = expr_to_var_sym(expr, &sym);
357         if (!name || !sym)
358                 goto free;
359         ret = get_sm_state(owner, name, sym);
360 free:
361         free_string(name);
362         return ret;
363 }

365 void delete_state(int owner, const char *name, struct symbol *sym)
366 {
367         delete_state_stree(&cur_stree, owner, name, sym);
368         if (cond_true_stack) {
369                 delete_state_stree_stack(&pre_cond_stack, owner, name, sym);
370                 delete_state_stree_stack(&cond_true_stack, owner, name, sym);
371                 delete_state_stree_stack(&cond_false_stack, owner, name, sym);
372         }
373 }

375 void delete_state_expr(int owner, struct expression *expr)
376 {
377         char *name;
378         struct symbol *sym;

380         expr = strip_expr(expr);
381         name = expr_to_var_sym(expr, &sym);
382         if (!name || !sym)
383                 goto free;
384         delete_state(owner, name, sym);
385 free:
386         free_string(name);
387 }

389 static void delete_all_states_stree_sym(struct stree **stree, struct symbol *sym
390 {
```

```
391         struct state_list *slist = NULL;
392         struct sm_state *sm;

394         FOR_EACH_SM(*stree, sm) {
395                 if (sm->sym == sym)
396                         add_ptr_list(&slist, sm);
397         } END_FOR_EACH_SM(sm);

399         FOR_EACH_PTR(slist, sm) {
400                 delete_state_stree(stree, sm->owner, sm->name, sm->sym);
401         } END_FOR_EACH_PTR(sm);

403         free_slist(&slist);
404 }

406 static void delete_all_states_stree_stack_sym(struct stree_stack **stack, struct
407 {
408         struct stree *stree;

410         if (!*stack)
411                 return;

413         stree = pop_stree(stack);
414         delete_all_states_stree_sym(&stree, sym);
415         push_stree(stack, stree);
416 }

418 void __delete_all_states_sym(struct symbol *sym)
419 {
420         delete_all_states_stree_sym(&cur_stree, sym);

422         delete_all_states_stree_stack_sym(&true_stack, sym);
423         delete_all_states_stree_stack_sym(&true_stack, sym);
424         delete_all_states_stree_stack_sym(&false_stack, sym);
425         delete_all_states_stree_stack_sym(&pre_cond_stack, sym);
426         delete_all_states_stree_stack_sym(&cond_true_stack, sym);
427         delete_all_states_stree_stack_sym(&cond_false_stack, sym);
428         delete_all_states_stree_stack_sym(&fake_cur_stree_stack, sym);
429         delete_all_states_stree_stack_sym(&break_stack, sym);
430         delete_all_states_stree_stack_sym(&fake_break_stack, sym);
431         delete_all_states_stree_stack_sym(&switch_stack, sym);
432         delete_all_states_stree_stack_sym(&continue_stack, sym);

434         /*
435          * deleting from the goto stack is problematic because we don't know
436          * if the label is in scope and also we need the value for --two-passes.
437          */
438 }

440 struct stree *get_all_states_from_stree(int owner, struct stree *source)
441 {
442         struct stree *ret = NULL;
443         struct sm_state *tmp;

445         FOR_EACH_SM(source, tmp) {
446                 if (tmp->owner == owner)
447                         avl_insert(&ret, tmp);
448         } END_FOR_EACH_SM(tmp);

450         return ret;
451 }

453 struct stree *get_all_states_stree(int owner)
454 {
455         return get_all_states_from_stree(owner, cur_stree);
456 }
```



```
458 struct stree *__get_cur_stree(void)
459 {
460         return cur_stree;
461 }

463 int is_reachable(void)
464 {
465         if (cur_stree)
466                 return 1;
467         return 0;
468 }

470 void set_true_false_states(int owner, const char *name, struct symbol *sym,
471                            struct smatch_state *true_state,
472                            struct smatch_state *false_state)
473 {
474         if (read_only)
475                 sm_perror("cur_stree is read only.");

477         if (option_debug || strcmp(check_name(owner), option_debug_check) == 0)
478                 struct smatch_state *tmp;

480                 tmp = get_state(owner, name, sym);
481                 sm_msg("%s [%s] '%s'.  Was %s.  Now T:%s F:%s", __func__,
482                        check_name(owner),  name, show_state(tmp),
483                        show_state(true_state), show_state(false_state));
484         }

486         if (unreachable())
487                 return;

489         if (!cond_false_stack || !cond_true_stack) {
490                 sm_perror("missing true/false stacks");
491                 return;
492         }

494         if (true_state)
495                 set_state_stree_stack(&cond_true_stack, owner, name, sym, true_s
496         if (false_state)
497                 set_state_stree_stack(&cond_false_stack, owner, name, sym, false
498 }

500 void set_true_false_states_expr(int owner, struct expression *expr,
501                                 struct smatch_state *true_state,
502                                 struct smatch_state *false_state)
503 {
504         char *name;
505         struct symbol *sym;

507         expr = strip_expr(expr);
508         name = expr_to_var_sym(expr, &sym);
509         if (!name || !sym)
510                 goto free;
511         set_true_false_states(owner, name, sym, true_state, false_state);
512 free:
513         free_string(name);
514 }

516 void __set_true_false_sm(struct sm_state *true_sm, struct sm_state *false_sm)
517 {
518         int owner;
519         const char *name;
520         struct symbol *sym;

522         if (!true_sm && !false_sm)
```

```
523                 return;

525         if (unreachable())
526                 return;

528         owner = true_sm ? true_sm->owner : false_sm->owner;
529         name = true_sm ? true_sm->name : false_sm->name;
530         sym = true_sm ? true_sm->sym : false_sm->sym;
531         if (option_debug || strcmp(check_name(owner), option_debug_check) == 0)
532                 struct smatch_state *tmp;

534                 tmp = get_state(owner, name, sym);
535                 sm_msg("%s [%s] '%s'.  Was %s.  Now T:%s F:%s", __func__,
536                        check_name(owner),  name, show_state(tmp),
537                        show_state(true_sm ? true_sm->state : NULL),
538                        show_state(false_sm ? false_sm->state : NULL));
539         }

541         if (!cond_false_stack || !cond_true_stack) {
542                 sm_perror("missing true/false stacks");
543                 return;
544         }

546         if (true_sm)
547                 overwrite_sm_state_stree_stack(&cond_true_stack, true_sm);
548         if (false_sm)
549                 overwrite_sm_state_stree_stack(&cond_false_stack, false_sm);
550 }

552 void nullify_path(void)
553 {
554         if (fake_cur_stree_stack) {
555                 __free_fake_cur_stree();
556                 __push_fake_cur_stree();
557         }
558         free_stree(&cur_stree);
559 }

561 void __match_nullify_path_hook(const char *fn, struct expression *expr,
562                                void *unused)
563 {
564         nullify_path();
565 }

567 /*
568  * At the start of every function we mark the path
569  * as unnull.  That way there is always at least one state
570  * in the cur_stree until nullify_path is called.  This
571  * is used in merge_slist() for the first null check.
572  */
573 void __unnullify_path(void)
574 {
575         if (!cur_stree)
576                 set_state(-1, "unnull_path", NULL, &true_state);
577 }

579 int __path_is_null(void)
580 {
581         if (cur_stree)
582                 return 0;
583         return 1;
584 }

586 static void check_stree_stack_free(struct stree_stack **stack)
587 {
588         if (*stack) {
```



```
589                 sm_perror("stack not empty");
590                 free_stack_and_strees(stack);
591         }
592 }

594 void save_all_states(void)
595 {
596         __add_ptr_list(&backup, cur_stree, 0);
597         cur_stree = NULL;

599         __add_ptr_list(&backup, true_stack, 0);
600         true_stack = NULL;
601         __add_ptr_list(&backup, false_stack, 0);
602         false_stack = NULL;
603         __add_ptr_list(&backup, pre_cond_stack, 0);
604         pre_cond_stack = NULL;

606         __add_ptr_list(&backup, cond_true_stack, 0);
607         cond_true_stack = NULL;
608         __add_ptr_list(&backup, cond_false_stack, 0);
609         cond_false_stack = NULL;

611         __add_ptr_list(&backup, fake_cur_stree_stack, 0);
612         fake_cur_stree_stack = NULL;

614         __add_ptr_list(&backup, break_stack, 0);
615         break_stack = NULL;
616         __add_ptr_list(&backup, fake_break_stack, 0);
617         fake_break_stack = NULL;

619         __add_ptr_list(&backup, switch_stack, 0);
620         switch_stack = NULL;
621         __add_ptr_list(&backup, remaining_cases, 0);
622         remaining_cases = NULL;
623         __add_ptr_list(&backup, default_stack, 0);
624         default_stack = NULL;
625         __add_ptr_list(&backup, continue_stack, 0);
626         continue_stack = NULL;

628         __add_ptr_list(&backup, goto_stack, 0);
629         goto_stack = NULL;
630 }

632 static void *pop_backup(void)
633 {
634         void *ret;

636         ret = last_ptr_list(backup);
637         delete_ptr_list_last(&backup);
638         return ret;
639 }

641 void restore_all_states(void)
642 {
643         goto_stack = pop_backup();

645         continue_stack = pop_backup();
646         default_stack = pop_backup();
647         remaining_cases = pop_backup();
648         switch_stack = pop_backup();
649         fake_break_stack = pop_backup();
650         break_stack = pop_backup();

652         fake_cur_stree_stack = pop_backup();

654         cond_false_stack = pop_backup();
```

```
655         cond_true_stack = pop_backup();

657         pre_cond_stack = pop_backup();
658         false_stack = pop_backup();
659         true_stack = pop_backup();

661         cur_stree = pop_backup();
662 }

664 void free_goto_stack(void)
665 {
666         struct named_stree *named_stree;

668         FOR_EACH_PTR(goto_stack, named_stree) {
669                 free_stree(&named_stree->stree);
670         } END_FOR_EACH_PTR(named_stree);
671         __free_ptr_list((struct ptr_list **)&goto_stack);
672 }

674 void clear_all_states(void)
675 {
676         nullify_path();
677         check_stree_stack_free(&true_stack);
678         check_stree_stack_free(&false_stack);
679         check_stree_stack_free(&pre_cond_stack);
680         check_stree_stack_free(&cond_true_stack);
681         check_stree_stack_free(&cond_false_stack);
682         check_stree_stack_free(&break_stack);
683         check_stree_stack_free(&fake_break_stack);
684         check_stree_stack_free(&switch_stack);
685         check_stree_stack_free(&continue_stack);
686         check_stree_stack_free(&fake_cur_stree_stack);

688         free_goto_stack();

690         free_every_single_sm_state();
691         free_tmp_expressions();
692 }

694 void __push_cond_stacks(void)
695 {
696         push_stree(&cond_true_stack, NULL);
697         push_stree(&cond_false_stack, NULL);
698         __push_fake_cur_stree();
699 }

701 void __fold_in_set_states(void)
702 {
703         struct stree *new_states;
704         struct sm_state *sm;

706         new_states = __pop_fake_cur_stree();
707         FOR_EACH_SM(new_states, sm) {
708                 __set_sm(sm);
709                 __set_true_false_sm(sm, sm);
710         } END_FOR_EACH_SM(sm);
711         free_stree(&new_states);
712 }

714 void __free_set_states(void)
715 {
716         struct stree *new_states;

718         new_states = __pop_fake_cur_stree();
719         free_stree(&new_states);
720 }
```



```
722 struct stree *__copy_cond_true_states(void)
723 {
724         struct stree *ret;

726         ret = pop_stree(&cond_true_stack);
727         push_stree(&cond_true_stack, clone_stree(ret));
728         return ret;
729 }

731 struct stree *__copy_cond_false_states(void)
732 {
733         struct stree *ret;

735         ret = pop_stree(&cond_false_stack);
736         push_stree(&cond_false_stack, clone_stree(ret));
737         return ret;
738 }

740 struct stree *__pop_cond_true_stack(void)
741 {
742         return pop_stree(&cond_true_stack);
743 }

745 struct stree *__pop_cond_false_stack(void)
746 {
747         return pop_stree(&cond_false_stack);
748 }

750 /*
751  * This combines the pre cond states with either the true or false states.
752  * For example:
753  * a = kmalloc() ; if (a !! foo(a)
754  * In the pre state a is possibly null.  In the true state it is non null.
755  * In the false state it is null.  Combine the pre and the false to get
756  * that when we call 'foo', 'a' is null.
757  */
758 static void __use_cond_stack(struct stree_stack **stack)
759 {
760         struct stree *stree;

762         free_stree(&cur_stree);

764         cur_stree = pop_stree(&pre_cond_stack);
765         push_stree(&pre_cond_stack, clone_stree(cur_stree));

767         stree = pop_stree(stack);
768         overwrite_stree(stree, &cur_stree);
769         push_stree(stack, stree);
770 }

772 void __use_pre_cond_states(void)
773 {
774         free_stree(&cur_stree);
775         cur_stree = pop_stree(&pre_cond_stack);
776 }

778 void __use_cond_true_states(void)
779 {
780         __use_cond_stack(&cond_true_stack);
781 }

783 void __use_cond_false_states(void)
784 {
785         __use_cond_stack(&cond_false_stack);
786 }
```

```
788 void __negate_cond_stacks(void)
789 {
790         struct stree *old_false, *old_true;

792         __use_cond_stack(&cond_false_stack);
793         old_false = pop_stree(&cond_false_stack);
794         old_true = pop_stree(&cond_true_stack);
795         push_stree(&cond_false_stack, old_true);
796         push_stree(&cond_true_stack, old_false);
797 }

799 void __and_cond_states(void)
800 {
801         and_stree_stack(&cond_true_stack);
802         or_stree_stack(&pre_cond_stack, cur_stree, &cond_false_stack);
803 }

805 void __or_cond_states(void)
806 {
807         or_stree_stack(&pre_cond_stack, cur_stree, &cond_true_stack);
808         and_stree_stack(&cond_false_stack);
809 }

811 void __save_pre_cond_states(void)
812 {
813         push_stree(&pre_cond_stack, clone_stree(cur_stree));
814 }

816 void __discard_pre_cond_states(void)
817 {
818         struct stree *tmp;

820         tmp = pop_stree(&pre_cond_stack);
821         free_stree(&tmp);
822 }

824 struct stree *__get_true_states(void)
825 {
826         return clone_stree(top_stree(cond_true_stack));
827 }

829 struct stree *__get_false_states(void)
830 {
831         return clone_stree(top_stree(cond_false_stack));
832 }

834 void __use_cond_states(void)
835 {
836         struct stree *pre, *pre_clone, *true_states, *false_states;

838         pre = pop_stree(&pre_cond_stack);
839         pre_clone = clone_stree(pre);

841         true_states = pop_stree(&cond_true_stack);
842         overwrite_stree(true_states, &pre);
843         free_stree(&true_states);
844         /* we use the true states right away */
845         free_stree(&cur_stree);
846         cur_stree = pre;

848         false_states = pop_stree(&cond_false_stack);
849         overwrite_stree(false_states, &pre_clone);
850         free_stree(&false_states);
851         push_stree(&false_stack, pre_clone);
852 }
```



```
854 void __push_true_states(void)
855 {
856         push_stree(&true_stack, clone_stree(cur_stree));
857 }

859 void __use_false_states(void)
860 {
861         free_stree(&cur_stree);
862         cur_stree = pop_stree(&false_stack);
863 }

865 void __discard_false_states(void)
866 {
867         struct stree *stree;

869         stree = pop_stree(&false_stack);
870         free_stree(&stree);
871 }

873 void __merge_false_states(void)
874 {
875         struct stree *stree;

877         stree = pop_stree(&false_stack);
878         merge_stree(&cur_stree, stree);
879         free_stree(&stree);
880 }

882 /*
883  * This function probably seemed common sensical when I wrote it but, oh wow,
884  * does it look subtle in retrospect.  Say we set a state on one side of the if
885  * else path but not on the other, then what we should record in the fake stree
886  * is the merged state.
887  *
888  * This function relies on the fact that the we always set the cur_stree as well
889  * and we already have the infrastructure to merge things correctly into the
890  * cur_stree.
891  *
892  * So instead of merging fake strees together which is probably a lot of work,
893  * we just use it as a list of set states and look up the actual current values
894  * in the cur_stree.
895  *
896  */
897 static void update_stree_with_merged(struct stree **stree)
898 {
899         struct state_list *slist = NULL;
900         struct sm_state *sm, *new;

902         FOR_EACH_SM(*stree, sm) {
903                 new = get_sm_state(sm->owner, sm->name, sm->sym);
904                 if (!new)  /* This can happen if we go out of scope */
905                         continue;
906                 add_ptr_list(&slist, new);
907         } END_FOR_EACH_SM(sm);

909         FOR_EACH_PTR(slist, sm) {
910                 overwrite_sm_state_stree(stree, sm);
911         } END_FOR_EACH_PTR(sm);

913         free_slist(&slist);
914 }

916 static void update_fake_stree_with_merged(void)
917 {
918         struct stree *stree;
```

```
920         if (!fake_cur_stree_stack)
921                 return;
922         stree = pop_stree(&fake_cur_stree_stack);
923         update_stree_with_merged(&stree);
924         push_stree(&fake_cur_stree_stack, stree);
925 }

927 void __merge_true_states(void)
928 {
929         struct stree *stree;

931         stree = pop_stree(&true_stack);
932         merge_stree(&cur_stree, stree);
933         update_fake_stree_with_merged();
934         free_stree(&stree);
935 }

937 void __push_continues(void)
938 {
939         push_stree(&continue_stack, NULL);
940 }

942 void __discard_continues(void)
943 {
944         struct stree *stree;

946         stree = pop_stree(&continue_stack);
947         free_stree(&stree);
948 }

950 void __process_continues(void)
951 {
952         struct stree *stree;

954         stree = pop_stree(&continue_stack);
955         if (!stree)
956                 stree = clone_stree(cur_stree);
957         else
958                 merge_stree(&stree, cur_stree);

960         push_stree(&continue_stack, stree);
961 }

963 void __merge_continues(void)
964 {
965         struct stree *stree;

967         stree = pop_stree(&continue_stack);
968         merge_stree(&cur_stree, stree);
969         free_stree(&stree);
970 }

972 void __push_breaks(void)
973 {
974         push_stree(&break_stack, NULL);
975         if (fake_cur_stree_stack)
976                 push_stree(&fake_break_stack, NULL);
977 }

979 void __process_breaks(void)
980 {
981         struct stree *stree;

983         stree = pop_stree(&break_stack);
984         if (!stree)
```



```
985                 stree = clone_stree(cur_stree);
986         else
987                 merge_stree(&stree, cur_stree);
988         push_stree(&break_stack, stree);

990         if (!fake_cur_stree_stack)
991                 return;

993         stree = pop_stree(&fake_break_stack);
994         if (!stree)
995                 stree = clone_stree(top_stree(fake_cur_stree_stack));
996         else
997                 merge_stree(&stree, top_stree(fake_cur_stree_stack));
998         push_stree(&fake_break_stack, stree);
999 }

1001 int __has_breaks(void)
1002 {
1003         struct stree *stree;
1004         int ret;

1006         stree = pop_stree(&break_stack);
1007         ret = !!stree;
1008         push_stree(&break_stack, stree);
1009         return ret;
1010 }

1012 void __merge_breaks(void)
1013 {
1014         struct stree *stree;
1015         struct sm_state *sm;

1017         stree = pop_stree(&break_stack);
1018         merge_stree(&cur_stree, stree);
1019         free_stree(&stree);

1021         if (!fake_cur_stree_stack)
1022                 return;

1024         stree = pop_stree(&fake_break_stack);
1025         update_stree_with_merged(&stree);
1026         FOR_EACH_SM(stree, sm) {
1027                 overwrite_sm_state_stree_stack(&fake_cur_stree_stack, sm);
1028         } END_FOR_EACH_SM(sm);
1029         free_stree(&stree);
1030 }

1032 void __use_breaks(void)
1033 {
1034         struct stree *stree;
1035         struct sm_state *sm;

1037         free_stree(&cur_stree);
1038         cur_stree = pop_stree(&break_stack);

1040         if (!fake_cur_stree_stack)
1041                 return;
1042         stree = pop_stree(&fake_break_stack);
1043         FOR_EACH_SM(stree, sm) {
1044                 overwrite_sm_state_stree_stack(&fake_cur_stree_stack, sm);
1045         } END_FOR_EACH_SM(sm);
1046         free_stree(&stree);


1049 }
```

```
1051 void __save_switch_states(struct expression *switch_expr)
1052 {
1053         struct range_list *rl;

1055         get_absolute_rl(switch_expr, &rl);

1057         push_rl(&remaining_cases, rl);
1058         push_stree(&switch_stack, clone_stree(cur_stree));
1059 }

1061 int have_remaining_cases(void)
1062 {
1063         return !!top_rl(remaining_cases);
1064 }

1066 void __merge_switches(struct expression *switch_expr, struct range_list *case_rl
1067 {
1068         struct stree *stree;
1069         struct stree *implied_stree;

1071         stree = pop_stree(&switch_stack);
1072         implied_stree = __implied_case_stree(switch_expr, case_rl, &remaining_ca
1073         merge_stree(&cur_stree, implied_stree);
1074         free_stree(&implied_stree);
1075         push_stree(&switch_stack, stree);
1076 }

1078 void __discard_switches(void)
1079 {
1080         struct stree *stree;

1082         pop_rl(&remaining_cases);
1083         stree = pop_stree(&switch_stack);
1084         free_stree(&stree);
1085 }

1087 void __push_default(void)
1088 {
1089         push_stree(&default_stack, NULL);
1090 }

1092 void __set_default(void)
1093 {
1094         set_state_stree_stack(&default_stack, 0, "has_default", NULL, &true_stat
1095 }

1097 int __pop_default(void)
1098 {
1099         struct stree *stree;

1101         stree = pop_stree(&default_stack);
1102         if (stree) {
1103                 free_stree(&stree);
1104                 return 1;
1105         }
1106         return 0;
1107 }

1109 static struct named_stree *alloc_named_stree(const char *name, struct symbol *sy
1110 {
1111         struct named_stree *named_stree = __alloc_named_stree(0);

1113         named_stree->name = (char *)name;
1114         named_stree->stree = stree;
1115         named_stree->sym = sym;
1116         return named_stree;
```



```
1117 }

1119 void __save_gotos(const char *name, struct symbol *sym)
1120 {
1121         struct stree **stree;
1122         struct stree *clone;

1124         stree = get_named_stree(goto_stack, name, sym);
1125         if (stree) {
1126                 merge_stree(stree, cur_stree);
1127                 return;
1128         } else {
1129                 struct named_stree *named_stree;

1131                 clone = clone_stree(cur_stree);
1132                 named_stree = alloc_named_stree(name, sym, clone);
1133                 add_ptr_list(&goto_stack, named_stree);
1134         }
1135 }

1137 void __merge_gotos(const char *name, struct symbol *sym)
1138 {
1139         struct stree **stree;

1141         stree = get_named_stree(goto_stack, name, sym);
1142         if (stree)
1143                 merge_stree(&cur_stree, *stree);
1144 }
```

```
**********************************************************
    7155 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_stored_conditions.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Keep a record of all the things we have tested for so that we know when we
 * test for it again.  For example, if we have code like this:
 *
 *      if (foo & FLAG)
 *              lock();
 *
 *      ...
 *
 *      if (foo & FLAG)
 *              unlock();
 *
 * That's the end goal at least.  But actually implementing the flow of that
 * requires quite a bit of work because if "foo" changes the condition needs to
 * be retested and smatch_implications.c needs to be updated.
 *
 * For now, I just record the conditions and use it to see if we test for NULL
 * twice.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"

static int my_id;
static int link_id;

static struct smatch_state *alloc_link_state(struct expression_list *expr_list)
{
        struct expression *tmp;
        struct smatch_state *state;
        static char buf[256];
        char *name;
        int i;

        state = __alloc_smatch_state(0);

        i = 0;
        FOR_EACH_PTR(expr_list, tmp) {
                name = expr_to_str(tmp);
                if (!i++) {
                        snprintf(buf, sizeof(buf), "%s", name);
                } else {
```



```
                        append(buf, ", ", sizeof(buf));
                        append(buf, name, sizeof(buf));
                }
                free_string(name);
        } END_FOR_EACH_PTR(tmp);

        state->name = alloc_sname(buf);
        state->data = expr_list;
        return state;
}

static struct expression_list *clone_expression_list(struct expression_list *lis
{
        struct expression_list *ret = NULL;
        struct expression *expr;

        FOR_EACH_PTR(list, expr) {
                add_ptr_list(&ret, expr);
        } END_FOR_EACH_PTR(expr);

        return ret;
}

static void insert_expression(struct expression_list **list, struct expression *
{
        struct expression *tmp;

        FOR_EACH_PTR(*list, tmp) {
                if (tmp == expr)
                        return;
        } END_FOR_EACH_PTR(tmp);

        add_ptr_list(list, expr);
}

static struct smatch_state *merge_links(struct smatch_state *s1, struct smatch_s
{
        struct expression_list *list, *expr_list;
        struct expression *expr;

        expr_list = clone_expression_list(s1->data);

        list = s2->data;
        FOR_EACH_PTR(list, expr) {
                insert_expression(&expr_list, expr);
        } END_FOR_EACH_PTR(expr);

        return alloc_link_state(expr_list);
}

static void save_link_var_sym(const char *var, struct symbol *sym, struct expres
{
        struct smatch_state *old_state, *new_state;
        struct expression_list *expr_list;

        old_state = get_state(link_id, var, sym);
        expr_list = clone_expression_list(old_state ? old_state->data : NULL);

        insert_expression(&expr_list, condition);

        new_state = alloc_link_state(expr_list);
        set_state(link_id, var, sym, new_state);
}

static void match_link_modify(struct sm_state *sm, struct expression *mod_expr)
{
```

```
127         struct expression_list *expr_list;
128         struct expression *tmp;
129         char *name;

131         expr_list = sm->state->data;

133         FOR_EACH_PTR(expr_list, tmp) {
134                 name = expr_to_str(tmp);
135                 set_state(my_id, name, NULL, &undefined);
136                 free_string(name);
137         } END_FOR_EACH_PTR(tmp);
138         set_state(link_id, sm->name, sm->sym, &undefined);
139 }

141 static struct smatch_state *alloc_state(struct expression *expr, int is_true)
142 {
143         struct smatch_state *state;

145         state = __alloc_smatch_state(0);
146         if (is_true)
147                 state->name = alloc_sname("true");
148         else
149                 state->name = alloc_sname("false");
150         state->data = expr;
151         return state;
152 }

154 static void store_all_links(struct expression *expr, struct expression *conditio
155 {
156         char *var;
157         struct symbol *sym;

159         expr = strip_expr(expr);

161         if (is_array(expr)) {
162                 var = expr_to_known_chunk_sym(expr, &sym);
163                 if (var)
164                         save_link_var_sym(var, sym, condition);
165         }

167         switch (expr->type) {
168         case EXPR_COMPARE:
169         case EXPR_BINOP:
170                 store_all_links(expr->left, condition);
171                 store_all_links(expr->right, condition);
172                 return;
173         case EXPR_VALUE:
174                 return;
175         }

177         var = expr_to_var_sym(expr, &sym);
178         if (!var || !sym)
179                 goto free;
180         save_link_var_sym(var, sym, condition);
181 free:
182         free_string(var);
183 }

185 static int condition_too_complicated(struct expression *expr)
186 {
187         if (get_complication_score(expr) > 2)
188                 return 1;
189         return 0;
190 }

192 void __stored_condition(struct expression *expr)
```



```
193 {
194         struct smatch_state *true_state, *false_state;
195         char *name;
196         sval_t val;

198         if (get_value(expr, &val))
199                 return;

201         if (condition_too_complicated(expr))
202                 return;

204         name = expr_to_str(expr);
205         if (!name)
206                 return;
207         true_state = alloc_state(expr, TRUE);
208         false_state = alloc_state(expr, FALSE);
209         set_true_false_states(my_id, name, NULL, true_state, false_state);
210         store_all_links(expr, expr);
211         free_string(name);
212 }

214 struct smatch_state *get_stored_condition(struct expression *expr)
215 {
216         struct smatch_state *state;
217         char *name;

219         name = expr_to_str(expr);
220         if (!name)
221                 return NULL;

223         state = get_state(my_id, name, NULL);
224         free_string(name);
225         return state;
226 }

228 struct expression_list *get_conditions(struct expression *expr)
229 {
230         struct smatch_state *state;

232         state = get_state_expr(link_id, expr);
233         if (!state)
234                 return NULL;
235         return state->data;
236 }

238 void register_stored_conditions(int id)
239 {
240         my_id = id;
241 }

243 void register_stored_conditions_links(int id)
244 {
245         link_id = id;
246         add_merge_hook(link_id, &merge_links);
247         add_modification_hook(link_id, &match_link_modify);
248 }

250 #define RECURSE_LIMIT 50

252 static void filter_by_sm(struct sm_state *sm,
253                          struct state_list **true_stack,
254                          struct state_list **false_stack,
255                          int *recurse_cnt)
256 {
257         if (!sm)
258                 return;
```

```
260         if ((*recurse_cnt)++ > RECURSE_LIMIT)
261                 return;

263         if (strcmp(sm->state->name, "true") == 0) {
264                 add_ptr_list(true_stack, sm);
265         } else if (strcmp(sm->state->name, "false") == 0) {
266                 add_ptr_list(false_stack, sm);
267         }

269         if (sm->merged) {
270                 filter_by_sm(sm->left, true_stack, false_stack, recurse_cnt);
271                 filter_by_sm(sm->right, true_stack, false_stack, recurse_cnt);
272         }
273 }

275 struct sm_state *stored_condition_implication_hook(struct expression *expr,
276                                 struct state_list **true_stack,
277                                 struct state_list **false_stack)
278 {
279         struct sm_state *sm;
280         char *name;
281         int recurse_cnt = 0;
282         struct state_list *tmp_true = NULL;
283         struct state_list *tmp_false = NULL;
284         struct sm_state *tmp;

286         expr = strip_expr(expr);

288         name = expr_to_str(expr);
289         if (!name)
290                 return NULL;

292         sm = get_sm_state(my_id, name, NULL);
293         free_string(name);
294         if (!sm)
295                 return NULL;
296         if (!sm->merged)
297                 return NULL;

299         filter_by_sm(sm, &tmp_true, &tmp_false, &recurse_cnt);
300         if (!tmp_true && !tmp_false)
301                 return NULL;
302         if (recurse_cnt > RECURSE_LIMIT) {
303                 sm = NULL;
304                 goto free;
305         }

307         FOR_EACH_PTR(tmp_true, tmp) {
308                 add_ptr_list(true_stack, tmp);
309         } END_FOR_EACH_PTR(tmp);

311         FOR_EACH_PTR(tmp_false, tmp) {
312                 add_ptr_list(false_stack, tmp);
313         } END_FOR_EACH_PTR(tmp);

315 free:
316         free_slist(&tmp_true);
317         free_slist(&tmp_false);
318         return sm;
319 }
```

```
**********************************************************
    1829 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_string_list.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

int list_has_string(struct string_list *str_list, const char *str)
{
        char *tmp;

        if (!str)
                return 0;

        FOR_EACH_PTR(str_list, tmp) {
                if (strcmp(tmp, str) < 0)
                        continue;
                if (strcmp(tmp, str) == 0)
                        return 1;
                return 0;
        } END_FOR_EACH_PTR(tmp);
        return 0;
}

void insert_string(struct string_list **str_list, const char *_new)
{
        char *new = (char *)_new;
        char *tmp;

        FOR_EACH_PTR(*str_list, tmp) {
                if (strcmp(tmp, new) < 0)
                        continue;
                else if (strcmp(tmp, new) == 0) {
                        return;
                } else {
                        INSERT_CURRENT(alloc_string(new), tmp);
                        return;
                }
        } END_FOR_EACH_PTR(tmp);
        new = alloc_string(new);
        add_ptr_list(str_list, new);
}

struct string_list *clone_str_list(struct string_list *orig)
{
        char *tmp;
        struct string_list *ret = NULL;
```



```
        FOR_EACH_PTR(orig, tmp) {
                add_ptr_list(&ret, tmp);
        } END_FOR_EACH_PTR(tmp);
        return ret;
}

struct string_list *combine_string_lists(struct string_list *one, struct string_
{
        struct string_list *ret;
        char *tmp;

        ret = clone_str_list(one);
        FOR_EACH_PTR(two, tmp) {
                insert_string(&ret, tmp);
        } END_FOR_EACH_PTR(tmp);
        return ret;
}
```

```
**********************************************************
    3821 Fri Dec 21 15:00:36 2018
new/usr/src/tools/smatch/src/smatch_strings.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2015 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static int get_str(void *_ret, int argc, char **argv, char **azColName)
{
        char **ret = _ret;

        if (*ret)
                *ret = (void *)-1UL;
        else
                *ret = alloc_sname(argv[0]);

        return 0;
}

static char *get_string_from_mtag(mtag_t tag)
{
        char *str = NULL;

        run_sql(get_str, &str,
                "select value from mtag_data where tag = %lld and offset = 0 and
                tag, STRING_VALUE);

        if ((unsigned long)str == -1UL)
                return NULL;
        return str;
}

struct expression *fake_string_from_mtag(mtag_t tag)
{
        char *str;

        if (!tag)
                return NULL;
        str = get_string_from_mtag(tag);
        if (!str)
                return NULL;
        return string_expression(str);
}
```



```
static void match_strcpy(const char *fn, struct expression *expr, void *unused)
{
        struct expression *dest, *src;

        dest = get_argument_from_call_expr(expr->args, 0);
        src = get_argument_from_call_expr(expr->args, 1);
        src = strip_expr(src);
        if (src->type == EXPR_STRING)
                set_state_expr(my_id, dest, alloc_state_str(src->string->data));
}

struct state_list *get_strings(struct expression *expr)
{
        struct state_list *ret = NULL;
        struct smatch_state *state;
        struct sm_state *sm;

        expr = strip_expr(expr);
        if (expr->type == EXPR_STRING) {
                state = alloc_state_str(expr->string->data);
                sm = alloc_sm_state(my_id, expr->string->data, NULL, state);
                add_ptr_list(&ret, sm);
                return ret;
        }

        if (expr->type == EXPR_CONDITIONAL ||
            expr->type == EXPR_SELECT) {
                struct state_list *true_strings = NULL;
                struct state_list *false_strings = NULL;

                if (known_condition_true(expr->conditional))
                        return get_strings(expr->cond_true);
                if (known_condition_false(expr->conditional))
                        return get_strings(expr->cond_false);

                true_strings = get_strings(expr->cond_true);
                false_strings = get_strings(expr->cond_false);
                concat_ptr_list((struct ptr_list *)true_strings, (struct ptr_lis
                free_slist(&true_strings);
                return false_strings;
        }

        sm = get_sm_state_expr(my_id, expr);
        if (!sm)
                return NULL;

        return clone_slist(sm->possible);
}

static void match_assignment(struct expression *expr)
{
        struct state_list *slist;
        struct sm_state *sm;

        if (expr->op != '=')
                return;

        slist = get_strings(strip_expr(expr->right));
        if (!slist)
                return;

        if (ptr_list_size((struct ptr_list *)slist) == 1) {
                sm = first_ptr_list((struct ptr_list *)slist);
                set_state_expr(my_id, expr->left, sm->state);
                return;
        }
```

```
127 }

129 static void match_string(struct expression *expr)
130 {
131         mtag_t tag;

133         if (expr->type != EXPR_STRING || !expr->string->data)
134                 return;
135         if (expr->string->length > 255)
136                 return;

138         if (!get_string_mtag(expr, &tag))
139                 return;

141         cache_sql(NULL, NULL, "insert into mtag_data values (%lld, %d, %d, '%q')
142                   tag, 0, STRING_VALUE, escape_newlines(expr->string->data));
143 }

145 void register_strings(int id)
146 {
147         my_id = id;

149         add_function_hook("strcpy", &match_strcpy, NULL);
150         add_function_hook("strlcpy", &match_strcpy, NULL);
151         add_function_hook("strncpy", &match_strcpy, NULL);

153         add_hook(&match_assignment, ASSIGNMENT_HOOK);
154         add_hook(&match_string, STRING_HOOK);

156 }
```

```
**********************************************************
    8868 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_strlen.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include <stdlib.h>
#include <errno.h>
#include "parse.h"
#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

#define UNKNOWN_SIZE (-1)

static int my_strlen_id;
/*
 * The trick with the my_equiv_id is that if we have:
 * foo = strlen(bar);
 * We don't know at that point what the strlen() is but we know it's equivalent
 * to "foo" so maybe we can find the value of "foo" later.
 */
static int my_equiv_id;

static struct smatch_state *size_to_estate(int size)
{
        sval_t sval;

        sval.type = &int_ctype;
        sval.value = size;

        return alloc_estate_sval(sval);
}

static struct smatch_state *unmatched_strlen_state(struct sm_state *sm)
{
        return size_to_estate(UNKNOWN_SIZE);
}

static void set_strlen_undefined(struct sm_state *sm, struct expression *mod_exp
{
        set_state(sm->owner, sm->name, sm->sym, size_to_estate(UNKNOWN_SIZE));
}

static void set_strlen_equiv_undefined(struct sm_state *sm, struct expression *m
{
        set_state(sm->owner, sm->name, sm->sym, &undefined);
}
```



```
static void match_string_assignment(struct expression *expr)
{
        struct range_list *rl;

        if (expr->op != '=')
                return;
        if (!get_implied_strlen(expr->right, &rl))
                return;
        set_state_expr(my_strlen_id, expr->left, alloc_estate_rl(clone_rl(rl)));
}

static void match_strlen(const char *fn, struct expression *expr, void *unused)
{
        struct expression *right;
        struct expression *str;
        struct expression *len_expr;
        char *len_name;
        struct smatch_state *state;

        right = strip_expr(expr->right);
        str = get_argument_from_call_expr(right->args, 0);
        len_expr = strip_expr(expr->left);

        len_name = expr_to_var(len_expr);
        if (!len_name)
                return;

        state = __alloc_smatch_state(0);
        state->name = len_name;
        state->data = len_expr;

        set_state_expr(my_equiv_id, str, state);
}

static void match_strlen_condition(struct expression *expr)
{
        struct expression *left;
        struct expression *right;
        struct expression *str = NULL;
        int strlen_left = 0;
        int strlen_right = 0;
        sval_t sval;
        struct smatch_state *true_state = NULL;
        struct smatch_state *false_state = NULL;
        int op;

        if (expr->type != EXPR_COMPARE)
                return;

        left = strip_expr(expr->left);
        right = strip_expr(expr->right);

        if (left->type == EXPR_CALL && sym_name_is("strlen", left->fn)) {
                str = get_argument_from_call_expr(left->args, 0);
                strlen_left = 1;
        }
        if (right->type == EXPR_CALL && sym_name_is("strlen", right->fn)) {
                str = get_argument_from_call_expr(right->args, 0);
                strlen_right = 1;
        }

        if (!strlen_left && !strlen_right)
                return;
        if (strlen_left && strlen_right)
                return;
```

```
127         op = expr->op;
128         if (strlen_left) {
129                 if (!get_value(right, &sval))
130                         return;
131         } else {
132                 op = flip_comparison(op);
133                 if (!get_value(left, &sval))
134                         return;
135         }

137         switch (op) {
138         case '<':
139         case SPECIAL_UNSIGNED_LT:
140                 true_state = size_to_estate(sval.value - 1);
141                 break;
142         case SPECIAL_LTE:
143         case SPECIAL_UNSIGNED_LTE:
144                 true_state = size_to_estate(sval.value);
145                 break;
146         case SPECIAL_EQUAL:
147                 true_state = size_to_estate(sval.value);
148                 break;
149         case SPECIAL_NOTEQUAL:
150                 false_state = size_to_estate(sval.value);
151                 break;
152         case SPECIAL_GTE:
153         case SPECIAL_UNSIGNED_GTE:
154                 false_state = size_to_estate(sval.value - 1);
155                 break;
156         case '>':
157         case SPECIAL_UNSIGNED_GT:
158                 false_state = size_to_estate(sval.value);
159                 break;
160         }

162         set_true_false_states_expr(my_strlen_id, str, true_state, false_state);
163 }

165 static void match_snprintf(const char *fn, struct expression *expr, void *unused
166 {
167         struct expression *dest;
168         struct expression *dest_size_expr;
169         sval_t limit_size;

171         dest = get_argument_from_call_expr(expr->args, 0);
172         dest_size_expr = get_argument_from_call_expr(expr->args, 1);

174         if (!get_implied_value(dest_size_expr, &limit_size))
175                 return;

177         if (limit_size.value <= 0)
178                 return;

180         set_state_expr(my_strlen_id, dest, size_to_estate(limit_size.value - 1))
181 }

183 static void match_strlcpycat(const char *fn, struct expression *expr, void *unus
184 {
185         struct expression *dest;
186         struct expression *src;
187         struct expression *limit_expr;
188         int src_len;
189         sval_t limit;

191         dest = get_argument_from_call_expr(expr->args, 0);
192         src = get_argument_from_call_expr(expr->args, 1);
```



```
193         limit_expr = get_argument_from_call_expr(expr->args, 2);

195         src_len = get_size_from_strlen(src);

197         if (!get_implied_max(limit_expr, &limit))
198                 return;
199         if (limit.value < 0 || limit.value > INT_MAX)
200                 return;
201         if (src_len != 0 && strcmp(fn, "strcpy") == 0 && src_len < limit.value)
202                 limit.value = src_len;

204         set_state_expr(my_strlen_id, dest, size_to_estate(limit.value - 1));
205 }

207 static void match_strcpy(const char *fn, struct expression *expr, void *unused)
208 {
209         struct expression *dest;
210         struct expression *src;
211         int src_len;

213         dest = get_argument_from_call_expr(expr->args, 0);
214         src = get_argument_from_call_expr(expr->args, 1);

216         src_len = get_size_from_strlen(src);
217         if (src_len == 0)
218                 return;

220         set_state_expr(my_strlen_id, dest, size_to_estate(src_len - 1));
221 }

223 static int get_strlen_from_string(struct expression *expr, struct range_list **r
224 {
225         sval_t sval;
226         int len;

228         len = expr->string->length;
229         sval = sval_type_val(&int_ctype, len - 1);
230         *rl = alloc_rl(sval, sval);
231         return 1;
232 }


235 static int get_strlen_from_state(struct expression *expr, struct range_list **rl
236 {
237         struct smatch_state *state;

239         state = get_state_expr(my_strlen_id, expr);
240         if (!state)
241                 return 0;
242         *rl = estate_rl(state);
243         return 1;
244 }

246 static int get_strlen_from_equiv(struct expression *expr, struct range_list **rl
247 {
248         struct smatch_state *state;

250         state = get_state_expr(my_equiv_id, expr);
251         if (!state || !state->data)
252                 return 0;
253         if (!get_implied_rl((struct expression *)state->data, rl))
254                 return 0;
255         return 1;
256 }

258 /*
```

```
 * This returns the strlen() without the NUL char.
 */
int get_implied_strlen(struct expression *expr, struct range_list **rl)
{

        *rl = NULL;

        expr = strip_expr(expr);
        if (expr->type == EXPR_STRING)
                return get_strlen_from_string(expr, rl);

        if (get_strlen_from_state(expr, rl))
                return 1;
        if (get_strlen_from_equiv(expr, rl))
                return 1;
        return 0;
}

int get_size_from_strlen(struct expression *expr)
{
        struct range_list *rl;
        sval_t max;

        if (!get_implied_strlen(expr, &rl))
                return 0;
        max = rl_max(rl);
        if (sval_is_negative(max) || sval_is_max(max))
                return 0;

        return max.value + 1; /* add one because strlen doesn't include the NULL
}

void set_param_strlen(const char *name, struct symbol *sym, char *key, char *val
{
        struct range_list *rl = NULL;
        struct smatch_state *state;
        char fullname[256];

        if (strncmp(key, "$", 1) != 0)
                return;

        snprintf(fullname, 256, "%s%s", name, key + 1);

        str_to_rl(&int_ctype, value, &rl);
        if (!rl || is_whole_rl(rl))
                return;
        state = alloc_estate_rl(rl);
        set_state(my_strlen_id, fullname, sym, state);
}

static void match_call(struct expression *expr)
{
        struct expression *arg;
        struct range_list *rl;
        int i;

        i = 0;
        FOR_EACH_PTR(expr->args, arg) {
                if (!get_implied_strlen(arg, &rl))
                        continue;
                if (!is_whole_rl(rl))
                        sql_insert_caller_info(expr, STR_LEN, i, "$", show_rl(rl
                i++;
        } END_FOR_EACH_PTR(arg);
}
```



```
static void struct_member_callback(struct expression *call, int param, char *pri
{
        if (sm->state == &merged)
                return;
        sql_insert_caller_info(call, STR_LEN, param, printed_name, sm->state->na
}

void register_strlen(int id)
{
        my_strlen_id = id;

        add_unmatched_state_hook(my_strlen_id, &unmatched_strlen_state);

        select_caller_info_hook(set_param_strlen, STR_LEN);
        add_hook(&match_string_assignment, ASSIGNMENT_HOOK);

        add_modification_hook(my_strlen_id, &set_strlen_undefined);
        add_merge_hook(my_strlen_id, &merge_estates);
        add_hook(&match_call, FUNCTION_CALL_HOOK);
        add_member_info_callback(my_strlen_id, struct_member_callback);
        add_hook(&match_strlen_condition, CONDITION_HOOK);

        add_function_hook("snprintf", &match_snprintf, NULL);

        add_function_hook("strlcpy", &match_strlcpycat, NULL);
        add_function_hook("strlcat", &match_strlcpycat, NULL);
        add_function_hook("strcpy", &match_strcpy, NULL);
}

void register_strlen_equiv(int id)
{
        my_equiv_id = id;
        add_function_assign_hook("strlen", &match_strlen, NULL);
        add_modification_hook(my_equiv_id, &set_strlen_equiv_undefined);
}
```

```
**********************************************************
   13392 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_struct_assignment.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * This file started out by saying that if you have:
 *
 *      struct foo one, two;
 *      ...
 *      one = two;
 *
 * That's equivalent to saying:
 *
 *      one.x = two.x;
 *      one.y = two.y;
 *
 * Turning an assignment like that into a bunch of small fake assignments is
 * really useful.
 *
 * The call to memcpy(&one, &two, sizeof(foo)); is the same as "one = two;" so
 * we can re-use the code.  And we may as well use it for memset() too.
 * Assigning pointers is almost the same:
 *
 *      p1 = p2;
 *
 * Is the same as:
 *
 *      p1->x = p2->x;
 *      p1->y = p2->y;
 *
 * The problem is that you can go a bit crazy with pointers to pointers.
 *
 *      p1->x->y->z->one->two->three = p2->x->y->z->one->two->three;
 *
 * I don't have a proper solution for this problem right now.  I just copy one
 * level and don't nest.  It should handle limitted nesting but intelligently.
 *
 * The other thing is that you end up with a lot of garbage assignments where
 * we record "x could be anything. x->y could be anything. x->y->z->a->b->c
 * could *also* be anything!".  There should be a better way to filter this
 * useless information.
 *
 */

#include "scope.h"
#include "smatch.h"
#include "smatch_slist.h"
```



```
#include "smatch_extra.h"

enum {
        COPY_NORMAL,
        COPY_MEMCPY,
        COPY_MEMSET,
};

static struct symbol *get_struct_type(struct expression *expr)
{
        struct symbol *type;

        type = get_type(expr);
        if (!type)
                return NULL;
        if (type->type == SYM_PTR) {
                type = get_real_base_type(type);
                if (!type)
                        return NULL;
        }
        if (type->type == SYM_STRUCT)
                return type;
        if (type->type == SYM_UNION)
                return type;
        return NULL;
}

static struct expression *get_right_base_expr(struct symbol *left_type, struct e
{
        struct symbol *struct_type;

        if (!right)
                return NULL;

        struct_type = get_struct_type(right);
        if (!struct_type)
                return NULL;
        if (struct_type != left_type)
                return NULL;

        if (right->type == EXPR_PREOP && right->op == '&')
                right = strip_expr(right->unop);

        if (right->type == EXPR_CALL)
                return NULL;

        if (is_pointer(right))
                right = deref_expression(right);

        return right;
}

static struct expression *remove_addr(struct expression *expr)
{
        struct symbol *type;

        expr = strip_expr(expr);
        if (!expr)
                return NULL;

        if (expr->type == EXPR_PREOP && expr->op == '&')
                return strip_expr(expr->unop);
        type = get_type(expr);
        if (!type)
                return expr;
        if (type->type != SYM_PTR && type->type != SYM_ARRAY)
```

```
127                 return expr;

129         return deref_expression(expr);
130 }

132 static struct expression *faked_expression;
133 struct expression *get_faked_expression(void)
134 {
135         if (!__in_fake_assign)
136                 return NULL;
137         return faked_expression;
138 }

140 static void split_fake_expr(struct expression *expr)
141 {
142         __in_fake_assign++;
143         __in_fake_struct_assign++;
144         __split_expr(expr);
145         __in_fake_struct_assign--;
146         __in_fake_assign--;
147 }

149 static void handle_non_struct_assignments(struct expression *left, struct expres
150 {
151         struct symbol *type;
152         struct expression *assign;

154         type = get_type(left);
155         if (!type)
156                 return;
157         if (type->type == SYM_PTR) {
158                 left = deref_expression(left);
159                 if (right)
160                         right = deref_expression(right);
161                 else
162                         right = unknown_value_expression(left);
163                 assign = assign_expression(left, '=', right);
164                 split_fake_expr(assign);
165                 return;
166         }
167         if (type->type != SYM_BASETYPE)
168                 return;
169         right = strip_expr(right);
170         if (!right)
171                 right = unknown_value_expression(left);
172         assign = assign_expression(left, '=', right);
173         split_fake_expr(assign);
174 }

176 static void set_inner_struct_members(int mode, struct expression *faked, struct
177 {
178         struct expression *left_member;
179         struct expression *right_member = NULL;  /* silence GCC */
180         struct expression *assign;
181         struct symbol *base = get_real_base_type(member);
182         struct symbol *tmp;

184         if (member->ident) {
185                 left = member_expression(left, '.', member->ident);
186                 if (mode != COPY_MEMSET && right)
187                         right = member_expression(right, '.', member->ident);
188         }

190         FOR_EACH_PTR(base->symbol_list, tmp) {
191                 struct symbol *type;
```



```
193                 type = get_real_base_type(tmp);
194                 if (!type)
195                         continue;

197                 if (type->type == SYM_ARRAY)
198                         continue;
199                 if (type->type == SYM_UNION || type->type == SYM_STRUCT) {
200                         set_inner_struct_members(mode, faked, left, right, tmp);
201                         continue;
202                 }
203                 if (!tmp->ident)
204                         continue;

206                 left_member = member_expression(left, '.', tmp->ident);

208                 switch (mode) {
209                 case COPY_NORMAL:
210                 case COPY_MEMCPY:
211                         if (right)
212                                 right_member = member_expression(right, '.', tmp
213                         else
214                                 right_member = unknown_value_expression(left_mem
215                         break;
216                 case COPY_MEMSET:
217                         right_member = right;
218                         break;
219                 }

221                 assign = assign_expression(left_member, '=', right_member);
222                 split_fake_expr(assign);
223         } END_FOR_EACH_PTR(tmp);
224 }

226 static void __struct_members_copy(int mode, struct expression *faked,
227                                   struct expression *left,
228                                   struct expression *right)
229 {
230         struct symbol *struct_type, *tmp, *type;
231         struct expression *left_member;
232         struct expression *right_member;
233         struct expression *assign;
234         int op = '.';


237         if (__in_fake_assign)
238                 return;
239         faked_expression = faked;

241         left = strip_expr(left);
242         right = strip_expr(right);

244         struct_type = get_struct_type(left);
245         if (!struct_type) {
246                 /*
247                  * This is not a struct assignment obviously.  But this is where
248                  * memcpy() is handled so it feels like a good place to add this
249                  * code.
250                  */
251                 handle_non_struct_assignments(left, right);
252                 goto done;
253         }

255         if (is_pointer(left)) {
256                 left = deref_expression(left);
257                 op = '*';
258         }
```

```
259         if (mode != COPY_MEMSET)
260                 right = get_right_base_expr(struct_type, right);
261
262         FOR_EACH_PTR(struct_type->symbol_list, tmp) {
263                 type = get_real_base_type(tmp);
264                 if (!type)
265                         continue;
266                 if (type->type == SYM_ARRAY)
267                         continue;
268
269                 if (type->type == SYM_UNION || type->type == SYM_STRUCT) {
270                         set_inner_struct_members(mode, faked, left, right, tmp);
271                         continue;
272                 }
273
274                 if (!tmp->ident)
275                         continue;
276
277                 left_member = member_expression(left, op, tmp->ident);
278                 right_member = NULL;
279
280                 switch (mode) {
281                 case COPY_NORMAL:
282                 case COPY_MEMCPY:
283                         if (right)
284                                 right_member = member_expression(right, op, tmp-
285                         else
286                                 right_member = unknown_value_expression(left_mem
287                         break;
288                 case COPY_MEMSET:
289                         right_member = right;
290                         break;
291                 }
292                 if (!right_member) {
293                         sm_perror("No right member");
294                         continue;
295                 }
296                 assign = assign_expression(left_member, '=', right_member);
297                 split_fake_expr(assign);
298         } END_FOR_EACH_PTR(tmp);
299
300 done:
301         faked_expression = NULL;
302 }
303
304 static int returns_zeroed_mem(struct expression *expr)
305 {
306         char *fn;
307
308         if (expr->type != EXPR_CALL || expr->fn->type != EXPR_SYMBOL)
309                 return 0;
310         fn = expr_to_var(expr->fn);
311         if (!fn)
312                 return 0;
313         if (strcmp(fn, "kcalloc") == 0)
314                 return 1;
315         if (option_project == PROJ_KERNEL && strstr(fn, "zalloc"))
316                 return 1;
317         return 0;
318 }
319
320 static int copy_containter_states(struct expression *left, struct expression *ri
321 {
322         char *left_name = NULL, *right_name = NULL;
323         struct symbol *left_sym, *right_sym;
324         struct sm_state *sm, *new_sm;
```



```
325         int ret = 0;
326         int len;
327         char buf[64];
328         char new_name[128];
329
330         right_name = expr_to_var_sym(right, &right_sym);
331         if (!right_name || !right_sym)
332                 goto free;
333         left_name = expr_to_var_sym(left, &left_sym);
334         if (!left_name || !left_sym)
335                 goto free;
336
337         len = snprintf(buf, sizeof(buf), "%s(-%d)", right_name, offset);
338         if (len >= sizeof(buf))
339                 goto free;
340
341         FOR_EACH_SM(__get_cur_stree(), sm) {
342                 if (sm->sym != right_sym)
343                         continue;
344                 if (strncmp(sm->name, buf, len) != 0)
345                         continue;
346                 snprintf(new_name, sizeof(new_name), "%s%s", left_name, sm->name
347                 new_sm = clone_sm(sm);
348                 new_sm->name = alloc_sname(new_name);
349                 new_sm->sym = left_sym;
350                 __set_sm(new_sm);
351                 ret = 1;
352         } END_FOR_EACH_SM(sm);
353 free:
354         free_string(left_name);
355         free_string(right_name);
356         return ret;
357 }
358
359 static int handle_param_offsets(struct expression *expr)
360 {
361         struct expression *right;
362         sval_t sval;
363
364         right = strip_expr(expr->right);
365
366         if (right->type != EXPR_BINOP || right->op != '-')
367                 return 0;
368
369         if (!get_value(right->right, &sval))
370                 return 0;
371
372         right = get_assigned_expr(right->left);
373         if (!right)
374                 return 0;
375         return copy_containter_states(expr->left, right, sval.value);
376 }
377
378 static void returns_container_of(struct expression *expr, int param, char *key,
379 {
380         struct expression *call, *arg;
381         int offset;
382
383         if (expr->type != EXPR_ASSIGNMENT || expr->op != '=')
384                 return;
385         call = strip_expr(expr->right);
386         if (call->type != EXPR_CALL)
387                 return;
388         if (param != -1)
389                 return;
390         param = atoi(key);
```

```
391         offset = atoi(value);

393         arg = get_argument_from_call_expr(call->args, param);
394         if (!arg)
395                 return;

397         copy_containter_states(expr->left, arg, -offset);
398 }

400 void __fake_struct_member_assignments(struct expression *expr)
401 {
402         struct symbol *left_type;

404         if (expr->op != '=')
405                 return;

407         if (is_zero(expr->right))
408                 return;

410         left_type = get_type(expr->left);
411         if (!left_type ||
412             (left_type->type != SYM_PTR &&
413              left_type->type != SYM_STRUCT))
414                 return;

416         if (handle_param_offsets(expr))
417                 return;

419         if (returns_zeroed_mem(expr->right))
420                 __struct_members_copy(COPY_MEMSET, expr, expr->left, zero_expr()
421         else
422                 __struct_members_copy(COPY_NORMAL, expr, expr->left, expr->right
423 }

425 static void match_memset(const char *fn, struct expression *expr, void *_size_ar
426 {
427         struct expression *buf;
428         struct expression *val;

430         buf = get_argument_from_call_expr(expr->args, 0);
431         val = get_argument_from_call_expr(expr->args, 1);

433         buf = strip_expr(buf);
434         __struct_members_copy(COPY_MEMSET, expr, remove_addr(buf), val);
435 }

437 static void match_memcpy(const char *fn, struct expression *expr, void *_arg)
438 {
439         struct expression *dest;
440         struct expression *src;

442         dest = get_argument_from_call_expr(expr->args, 0);
443         src = get_argument_from_call_expr(expr->args, 1);

445         __struct_members_copy(COPY_MEMCPY, expr, remove_addr(dest), remove_addr(
446 }

448 static void match_memcpy_unknown(const char *fn, struct expression *expr, void *
449 {
450         struct expression *dest;

452         dest = get_argument_from_call_expr(expr->args, 0);
453         __struct_members_copy(COPY_MEMCPY, expr, remove_addr(dest), NULL);
454 }

456 static void match_sscanf(const char *fn, struct expression *expr, void *unused)
```



```
457 {
458         struct expression *arg;
459         int i;

461         i = -1;
462         FOR_EACH_PTR(expr->args, arg) {
463                 if (++i < 2)
464                         continue;
465                 __struct_members_copy(COPY_MEMCPY, expr, remove_addr(arg), NULL)
466         } END_FOR_EACH_PTR(arg);
467 }

469 static void unop_expr(struct expression *expr)
470 {
471         if (expr->op != SPECIAL_INCREMENT &&
472             expr->op != SPECIAL_DECREMENT)
473                 return;

475         if (!is_pointer(expr))
476                 return;
477         faked_expression = expr;
478         __struct_members_copy(COPY_MEMCPY, expr, expr->unop, NULL);
479         faked_expression = NULL;
480 }

482 static void register_clears_param(void)
483 {
484         struct token *token;
485         char name[256];
486         const char *function;
487         int param;

489         if (option_project == PROJ_NONE)
490                 return;

492         snprintf(name, 256, "%s.clears_argument", option_project_str);

494         token = get_tokens_file(name);
495         if (!token)
496                 return;
497         if (token_type(token) != TOKEN_STREAMBEGIN)
498                 return;
499         token = token->next;
500         while (token_type(token) != TOKEN_STREAMEND) {
501                 if (token_type(token) != TOKEN_IDENT)
502                         return;
503                 function = show_ident(token->ident);
504                 token = token->next;
505                 if (token_type(token) != TOKEN_NUMBER)
506                         return;
507                 param = atoi(token->number);
508                 add_function_hook(function, &match_memcpy_unknown, INT_PTR(param
509                 token = token->next;
510         }
511         clear_token_alloc();
512 }

514 static void db_param_cleared(struct expression *expr, int param, char *key, char
515 {
516         struct expression *arg;

518         while (expr->type == EXPR_ASSIGNMENT)
519                 expr = strip_expr(expr->right);
520         if (expr->type != EXPR_CALL)
521                 return;
```

```
523         /*
524          * FIXME:  __struct_members_copy() requires an expression but
525          * get_variable_from_key() returns a name/sym pair so that doesn't
526          * work here.
527          */
528         if (strcmp(key, "$") != 0)
529                 return;

531         arg = get_argument_from_call_expr(expr->args, param);
532         if (!arg)
533                 return;

535         if (strcmp(value, "0") == 0)
536                 __struct_members_copy(COPY_MEMSET, expr, remove_addr(arg), zero_
537         else
538                 __struct_members_copy(COPY_MEMCPY, expr, remove_addr(arg), NULL)
539 }

541 void register_struct_assignment(int id)
542 {
543         add_function_hook("memset", &match_memset, NULL);
544         add_function_hook("__memset", &match_memset, NULL);

546         add_function_hook("memcpy", &match_memcpy, INT_PTR(0));
547         add_function_hook("memmove", &match_memcpy, INT_PTR(0));
548         add_function_hook("__memcpy", &match_memcpy, INT_PTR(0));
549         add_function_hook("__memmove", &match_memcpy, INT_PTR(0));

551         add_function_hook("sscanf", &match_sscanf, NULL);

553         add_hook(&unop_expr, OP_HOOK);
554         register_clears_param();
555         select_return_states_hook(PARAM_CLEARED, &db_param_cleared);

557         select_return_states_hook(CONTAINER, &returns_container_of);
558 }
```

```
**********************************************************
   13830 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_sval.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2012 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Basically the point of sval is that it can hold both ULLONG_MAX and
 * LLONG_MIN.  If it is an unsigned type then we use sval.uvalue or if it is
 * signed we use sval.value.
 *
 * I considered just using one bit to store whether the value was signed vs
 * unsigned but I think it might help to have the type information so we know
 * how to do type promotion.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

__ALLOCATOR(sval_t, "svals", sval);

sval_t *sval_alloc(sval_t sval)
{
        sval_t *ret;

        ret = __alloc_sval(0);
        *ret = sval;
        return ret;
}

sval_t *sval_alloc_permanent(sval_t sval)
{
        sval_t *ret;

        ret = malloc(sizeof(*ret));
        *ret = sval;
        return ret;
}

sval_t sval_blank(struct expression *expr)
{
        sval_t ret;

        ret.type = get_type(expr);
        if (!ret.type)
                ret.type = &int_ctype;
        ret.value = 123456789;
```



```
        return ret;
}

sval_t sval_type_val(struct symbol *type, long long val)
{
        sval_t ret;

        if (!type)
                type = &int_ctype;

        ret.type = type;
        ret.value = val;
        return ret;
}

sval_t sval_from_val(struct expression *expr, long long val)
{
        sval_t ret;

        ret = sval_blank(expr);
        ret.value = val;
        ret = sval_cast(get_type(expr), ret);

        return ret;
}

int sval_is_ptr(sval_t sval)
{
        if (!sval.type)
                return 0;
        return (sval.type->type == SYM_PTR || sval.type->type == SYM_ARRAY);
}

int sval_unsigned(sval_t sval)
{
        return type_unsigned(sval.type);
}

int sval_signed(sval_t sval)
{
        return !type_unsigned(sval.type);
}

int sval_bits(sval_t sval)
{
        return type_bits(sval.type);
}

int sval_bits_used(sval_t sval)
{
        int i;

        for (i = 64; i >= 1; i--) {
                if (sval.uvalue & (1ULL << (i - 1)))
                        return i;
        }
        return 0;
}

int sval_is_negative(sval_t sval)
{
        if (type_unsigned(sval.type))
                return 0;
        if (sval.value < 0)
                return 1;
```

```
127         return 0;
128 }

130 int sval_is_positive(sval_t sval)
131 {
132         return !sval_is_negative(sval);
133 }

135 int sval_is_min(sval_t sval)
136 {
137         sval_t min = sval_type_min(sval.type);

139         if (sval_unsigned(sval)) {
140                 if (sval.uvalue == 0)
141                         return 1;
142                 return 0;
143         }
144         /* return true for less than min as well */
145         return (sval.value <= min.value);
146 }

148 int sval_is_max(sval_t sval)
149 {
150         sval_t max = sval_type_max(sval.type);

152         if (sval_unsigned(sval))
153                 return (sval.uvalue >= max.value);
154         return (sval.value >= max.value);
155 }

157 int sval_is_a_min(sval_t sval)
158 {
159         if (sval_is_min(sval))
160                 return 1;
161         if (sval_signed(sval) && sval.value == SHRT_MIN)
162                 return 1;
163         if (sval_signed(sval) && sval.value == INT_MIN)
164                 return 1;
165         if (sval_signed(sval) && sval.value == LLONG_MIN)
166                 return 1;
167         return 0;
168 }

170 int sval_is_a_max(sval_t sval)
171 {
172         if (sval_is_max(sval))
173                 return 1;
174         if (sval.uvalue == SHRT_MAX)
175                 return 1;
176         if (sval.uvalue == INT_MAX)
177                 return 1;
178         if (sval.uvalue == LLONG_MAX)
179                 return 1;
180         if (sval.uvalue == USHRT_MAX)
181                 return 1;
182         if (sval.uvalue == UINT_MAX)
183                 return 1;
184         if (sval_unsigned(sval) && sval.uvalue == ULLONG_MAX)
185                 return 1;
186         if (sval.value > valid_ptr_max - 1000 &&
187             sval.value < valid_ptr_max + 1000)
188                 return 1;
189         return 0;
190 }

192 int sval_is_negative_min(sval_t sval)
```



```
193 {
194         if (!sval_is_negative(sval))
195                 return 0;
196         return sval_is_min(sval);
197 }

199 int sval_cmp_t(struct symbol *type, sval_t one, sval_t two)
200 {
201         sval_t one_cast, two_cast;

203         one_cast = sval_cast(type, one);
204         two_cast = sval_cast(type, two);
205         return sval_cmp(one_cast, two_cast);
206 }

208 int sval_cmp_val(sval_t one, long long val)
209 {
210         sval_t sval;

212         sval = sval_type_val(&llong_ctype, val);
213         return sval_cmp(one, sval);
214 }

216 sval_t sval_min(sval_t one, sval_t two)
217 {
218         if (sval_cmp(one, two) > 0)
219                 return two;
220         return one;
221 }

223 sval_t sval_max(sval_t one, sval_t two)
224 {
225         if (sval_cmp(one, two) < 0)
226                 return two;
227         return one;
228 }

230 int sval_too_low(struct symbol *type, sval_t sval)
231 {
232         if (sval_is_negative(sval) && type_unsigned(type))
233                 return 1;
234         if (type_signed(type) &&  sval_unsigned(sval))
235                 return 0;
236         if (type_signed(sval.type) &&
237             sval.value < sval_type_min(type).value)
238                 return 1;
239         if (sval_cmp(sval, sval_type_min(type)) < 0)
240                 return 1;
241         return 0;
242 }

244 int sval_too_high(struct symbol *type, sval_t sval)
245 {
246         if (sval_is_negative(sval))
247                 return 0;
248         if (sval.uvalue > sval_type_max(type).uvalue)
249                 return 1;
250         return 0;
251 }

253 int sval_fits(struct symbol *type, sval_t sval)
254 {
255         if (sval_too_low(type, sval))
256                 return 0;
257         if (sval_too_high(type, sval))
258                 return 0;
```

```
259         return 1;
260 }

262 sval_t sval_cast(struct symbol *type, sval_t sval)
263 {
264         sval_t ret;

266         if (!type)
267                 type = &int_ctype;

269         ret.type = type;
270         switch (sval_bits(ret)) {
271         case 1:
272                 ret.value = !!sval.value;
273                 break;
274         case 8:
275                 if (sval_unsigned(ret))
276                         ret.value = (long long)(unsigned char)sval.value;
277                 else
278                         ret.value = (long long)(char)sval.value;
279                 break;
280         case 16:
281                 if (sval_unsigned(ret))
282                         ret.value = (long long)(unsigned short)sval.value;
283                 else
284                         ret.value = (long long)(short)sval.value;
285                 break;
286         case 32:
287                 if (sval_unsigned(ret))
288                         ret.value = (long long)(unsigned int)sval.value;
289                 else
290                         ret.value = (long long)(int)sval.value;
291                 break;
292         default:
293                 ret.value = sval.value;
294         }
295         return ret;

297 }

299 sval_t sval_preop(sval_t sval, int op)
300 {
301         switch (op) {
302         case '!':
303                 sval.value = !sval.value;
304                 break;
305         case '~':
306                 sval.value = ~sval.value;
307                 sval = sval_cast(sval.type, sval);
308                 break;
309         case '-':
310                 sval.value = -sval.value;
311                 sval = sval_cast(sval.type, sval);
312                 break;
313         }
314         return sval;
315 }

317 static sval_t sval_binop_unsigned(struct symbol *type, sval_t left, int op, sval
318 {
319         sval_t ret;

321         ret.type = type;
322         switch (op) {
323         case '*':
324                 ret.uvalue =  left.uvalue * right.uvalue;
```



```
325                 break;
326         case '/':
327                 if (right.uvalue == 0) {
328                         sm_debug("%s: divide by zero", __func__);
329                         ret.uvalue = 123456789;
330                 } else {
331                         ret.uvalue = left.uvalue / right.uvalue;
332                 }
333                 break;
334         case '+':
335                 ret.uvalue = left.uvalue + right.uvalue;
336                 break;
337         case '-':
338                 ret.uvalue = left.uvalue - right.uvalue;
339                 break;
340         case '%':
341                 if (right.uvalue == 0) {
342                         sm_perror(" %s: MOD by zero", __func__);
343                         ret.uvalue = 123456789;
344                 } else {
345                         ret.uvalue = left.uvalue % right.uvalue;
346                 }
347                 break;
348         case '|':
349                 ret.uvalue = left.uvalue | right.uvalue;
350                 break;
351         case '&':
352                 ret.uvalue = left.uvalue & right.uvalue;
353                 break;
354         case SPECIAL_RIGHTSHIFT:
355                 ret.uvalue = left.uvalue >> right.uvalue;
356                 break;
357         case SPECIAL_LEFTSHIFT:
358                 ret.uvalue = left.uvalue << right.uvalue;
359                 break;
360         case '^':
361                 ret.uvalue = left.uvalue ^ right.uvalue;
362                 break;
363         default:
364                 sm_perror(" %s: unhandled binop %s", __func__,
365                         show_special(op));
366                 ret.uvalue = 1234567;
367         }
368         return ret;
369 }


372 static sval_t sval_binop_signed(struct symbol *type, sval_t left, int op, sval_t
373 {
374         sval_t ret;

376         ret.type = type;
377         switch (op) {
378         case '*':
379                 ret.value =  left.value * right.value;
380                 break;
381         case '/':
382                 if (right.value == 0) {
383                         sm_debug("%s: divide by zero", __func__);
384                         ret.value = 123456789;
385                 } else if (left.value == LLONG_MIN && right.value == -1) {
386                         sm_debug("%s: invalid divide LLONG_MIN/-1", __func__);
387                         ret.value = 12345678;
388                 } else {
389                         ret.value = left.value / right.value;
390                 }
```

```
391                 break;
392         case '+':
393                 ret.value = left.value + right.value;
394                 break;
395         case '-':
396                 ret.value = left.value - right.value;
397                 break;
398         case '%':
399                 if (right.value == 0) {
400                         sm_perror(" %s: MOD by zero", __func__);
401                         ret.value = 123456789;
402                 } else {
403                         ret.value = left.value % right.value;
404                 }
405                 break;
406         case '|':
407                 ret.value = left.value | right.value;
408                 break;
409         case '&':
410                 ret.value = left.value & right.value;
411                 break;
412         case SPECIAL_RIGHTSHIFT:
413                 ret.value = left.value >> right.value;
414                 break;
415         case SPECIAL_LEFTSHIFT:
416                 ret.value = left.value << right.value;
417                 break;
418         case '^':
419                 ret.value = left.value ^ right.value;
420                 break;
421         default:
422                 sm_perror(" %s: unhandled binop %s", __func__,
423                         show_special(op));
424                 ret.value = 1234567;
425         }
426         return ret;
427 }

429 static sval_t ptr_binop(struct symbol *type, sval_t left, int op, sval_t right)
430 {
431         sval_t ret;
432         int align;

434         if (op != '+' && op != '-')
435                 return sval_binop_unsigned(type, left, op, right);

437         ret.type = type;
438         if (type->type == SYM_PTR)
439                 type = get_real_base_type(type);
440         align = type->ctype.alignment;
441         if (align <= 0)
442                 align = 1;

444         if (op == '+') {
445                 if (type_is_ptr(left.type))
446                         ret.value = left.value + right.value * align;
447                 else
448                         ret.value = left.value * align + right.value;
449         } else {
450                 if (!type_is_ptr(left.type)) {
451                         left.value = -left.value;
452                         ret = ptr_binop(type, left, '+', right);
453                 } else if (!type_is_ptr(right.type)) {
454                         right.value = -right.value;
455                         ret = ptr_binop(type, left, '+', right);
456                 } else {
```



```
457                         ret.value = (left.value - right.value) / align;
458                 }
459         }

461         return ret;
462 }

464 sval_t sval_binop(sval_t left, int op, sval_t right)
465 {
466         struct symbol *type;
467         sval_t ret;

469         type = get_promoted_type(left.type, right.type);

471         if (type_is_ptr(type))
472                 ret = ptr_binop(type, left, op, right);
473         else if (type_unsigned(type))
474                 ret = sval_binop_unsigned(type, left, op, right);
475         else
476                 ret = sval_binop_signed(type, left, op, right);
477         return sval_cast(type, ret);
478 }

480 int sval_unop_overflows(sval_t sval, int op)
481 {
482         if (op != '-')
483                 return 0;
484         if (sval_positive_bits(sval) == 32 && sval.value == INT_MIN)
485                 return 1;
486         if (sval_positive_bits(sval) == 64 && sval.value == LLONG_MIN)
487                 return 1;
488         if (sval_is_negative(sval))
489                 return 0;
490         if (sval_signed(sval))
491                 return 0;
492         if (sval_bits(sval) == 32 && sval.uvalue > INT_MAX)
493                 return 1;
494         if (sval_bits(sval) == 64 && sval.uvalue > LLONG_MAX)
495                 return 1;
496         return 0;
497 }

499 int sval_binop_overflows(sval_t left, int op, sval_t right)
500 {
501         struct symbol *type;
502         sval_t max, min;

504         type = left.type;
505         if (type_positive_bits(right.type) > type_positive_bits(left.type))
506                 type = right.type;
507         if (type_positive_bits(type) < 31)
508                 type = &int_ctype;

510         max = sval_type_max(type);
511         min = sval_type_min(type);

513         switch (op) {
514         case '+':
515                 if (sval_is_negative(left) && sval_is_negative(right)) {
516                         if (left.value < min.value + right.value)
517                                 return 1;
518                         return 0;
519                 }
520                 if (sval_is_negative(left) || sval_is_negative(right))
521                         return 0;
522                 if (left.uvalue > max.uvalue - right.uvalue)
```

```
523                                 return 1;
524                 return 0;
525         case '*':
526                 if (type_signed(type)) {
527                         if (left.value == 0 || right.value == 0)
528                                 return 0;
529                         if (left.value > max.value / right.value)
530                                 return 1;
531                         if (left.value == -1 || right.value == -1)
532                                 return 0;
533                         return left.value != left.value * right.value / right.va

535                 }
536                 return right.uvalue != 0 && left.uvalue > max.uvalue / right.uva
537         case '-':
538                 if (type_unsigned(type)) {
539                         if (sval_cmp(left, right) < 0)
540                                 return 1;
541                         return 0;
542                 }
543                 if (sval_is_negative(left) && sval_is_negative(right))
544                         return 0;

546                 if (sval_is_negative(left)) {
547                         if (left.value < min.value + right.value)
548                                 return 1;
549                         return 0;
550                 }
551                 if (sval_is_negative(right)) {
552                         if (right.value == min.value)
553                                 return 1;
554                         right = sval_preop(right, '-');
555                         if (sval_binop_overflows(left, '+', right))
556                                 return 1;
557                         return 0;
558                 }
559                 return 0;
560         case SPECIAL_LEFTSHIFT:
561                 if (sval_cmp(left, sval_binop(max, invert_op(op), right)) > 0)
562                         return 1;
563                 return 0;
564         }
565         return 0;
566 }

568 int sval_binop_overflows_no_sign(sval_t left, int op, sval_t right)
569 {

571         struct symbol *type;

573         type = left.type;
574         if (type_positive_bits(right.type) > type_positive_bits(left.type))
575                 type = right.type;
576         if (type_positive_bits(type) <= 31)
577                 type = &uint_ctype;
578         else
579                 type = &ullong_ctype;

581         left = sval_cast(type, left);
582         right = sval_cast(type, right);
583         return sval_binop_overflows(left, op, right);
584 }

586 unsigned long long fls_mask(unsigned long long uvalue)
587 {
588         unsigned long long high_bit = 0;
```



```
590         while (uvalue) {
591                 uvalue >>= 1;
592                 high_bit++;
593         }

595         if (high_bit == 0)
596                 return 0;

598         return ((unsigned long long)-1) >> (64 - high_bit);
599 }

601 unsigned long long sval_fls_mask(sval_t sval)
602 {
603         return fls_mask(sval.uvalue);
604 }

606 const char *sval_to_str(sval_t sval)
607 {
608         char buf[30];

610         if (sval_unsigned(sval) && sval.value == ULLONG_MAX)
611                 return "u64max";
612         if (sval_unsigned(sval) && sval.value == UINT_MAX)
613                 return "u32max";
614         if (sval.value == USHRT_MAX)
615                 return "u16max";

617         if (sval_signed(sval) && sval.value == LLONG_MAX)
618                 return "s64max";
619         if (sval.value == INT_MAX)
620                 return "s32max";
621         if (sval.value == SHRT_MAX)
622                 return "s16max";

624         if (sval_signed(sval) && sval.value == SHRT_MIN)
625                 return "s16min";
626         if (sval_signed(sval) && sval.value == INT_MIN)
627                 return "s32min";
628         if (sval_signed(sval) && sval.value == LLONG_MIN)
629                 return "s64min";

631         if (sval_unsigned(sval))
632                 snprintf(buf, sizeof(buf), "%llu", sval.value);
633         else if (sval.value < 0)
634                 snprintf(buf, sizeof(buf), "(%lld)", sval.value);
635         else
636                 snprintf(buf, sizeof(buf), "%lld", sval.value);

638         return alloc_sname(buf);
639 }

641 const char *sval_to_numstr(sval_t sval)
642 {
643         char buf[30];

645         if (sval_unsigned(sval))
646                 snprintf(buf, sizeof(buf), "%llu", sval.value);
647         else if (sval.value < 0)
648                 snprintf(buf, sizeof(buf), "(%lld)", sval.value);
649         else
650                 snprintf(buf, sizeof(buf), "%lld", sval.value);

652         return alloc_sname(buf);
653 }
```

```
655 sval_t ll_to_sval(long long val)
656 {
657         sval_t ret;

659         ret.type = &llong_ctype;
660         ret.value = val;
661         return ret;
662 }

664 static void free_svals(struct symbol *sym)
665 {
666         if (__inline_fn)
667                 return;
668         clear_sval_alloc();
669 }

671 void register_sval(int my_id)
672 {
673         add_hook(&free_svals, AFTER_FUNC_HOOK);
674 }
```

```
**********************************************************
    2461 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_tracker.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

ALLOCATOR(tracker, "trackers");

struct tracker *alloc_tracker(int owner, const char *name, struct symbol *sym)
{
        struct tracker *tmp;

        tmp = __alloc_tracker(0);
        tmp->name = alloc_string(name);
        tmp->owner = owner;
        tmp->sym = sym;
        return tmp;
}

void add_tracker(struct tracker_list **list, int owner, const char *name,
                 struct symbol *sym)
{
        struct tracker *tmp;

        if (in_tracker_list(*list, owner, name, sym))
                return;
        tmp = alloc_tracker(owner, name, sym);
        add_ptr_list(list, tmp);
}

void add_tracker_expr(struct tracker_list **list, int owner, struct expression *
{
        char *name;
        struct symbol *sym;

        name = expr_to_var_sym(expr, &sym);
        if (!name || !sym)
                goto free;
        add_tracker(list, owner, name, sym);
free:
        free_string(name);
}

static void free_tracker(struct tracker *t)
{
        free_string(t->name);
        __free_tracker(t);
```



```
}

void del_tracker(struct tracker_list **list, int owner, const char *name,
                 struct symbol *sym)
{
        struct tracker *tmp;

        FOR_EACH_PTR(*list, tmp) {
                if (tmp->owner == owner && tmp->sym == sym
                    && !strcmp(tmp->name, name)) {
                        DELETE_CURRENT_PTR(tmp);
                        free_tracker(tmp);
                        return;
                }
        } END_FOR_EACH_PTR(tmp);
}

int in_tracker_list(struct tracker_list *list, int owner, const char *name,
                 struct symbol *sym)
{
        struct tracker *tmp;

        FOR_EACH_PTR(list, tmp) {
                if (tmp->owner == owner && tmp->sym == sym
                    && !strcmp(tmp->name, name))
                        return 1;
        } END_FOR_EACH_PTR(tmp);
        return 0;
}

void free_tracker_list(struct tracker_list **list)
{
        __free_ptr_list((struct ptr_list **)list);
}

void free_trackers_and_list(struct tracker_list **list)
{
        struct tracker *tmp;

        FOR_EACH_PTR(*list, tmp) {
                free_tracker(tmp);
        } END_FOR_EACH_PTR(tmp);
        free_tracker_list(list);
}
```

```
**********************************************************
   16959 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2009 Dan Carpenter.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The idea here is that you have an expression and you
 * want to know what the type is for that.
 */

#include "smatch.h"
#include "smatch_slist.h"

struct symbol *get_real_base_type(struct symbol *sym)
{
        struct symbol *ret;

        if (!sym)
                return NULL;
        ret = get_base_type(sym);
        if (!ret)
                return NULL;
        if (ret->type == SYM_RESTRICT || ret->type == SYM_NODE)
                return get_real_base_type(ret);
        return ret;
}

int type_bytes(struct symbol *type)
{
        int bits;

        if (type && type->type == SYM_ARRAY)
                return array_bytes(type);

        bits = type_bits(type);
        if (bits < 0)
                return 0;
        return bits_to_bytes(bits);
}

int array_bytes(struct symbol *type)
{
        if (!type || type->type != SYM_ARRAY)
                return 0;
        return bits_to_bytes(type->bit_size);
}

static struct symbol *get_binop_type(struct expression *expr)
```



```
{
        struct symbol *left, *right;

        left = get_type(expr->left);
        if (!left)
                return NULL;

        if (expr->op == SPECIAL_LEFTSHIFT ||
            expr->op == SPECIAL_RIGHTSHIFT) {
                if (type_positive_bits(left) < 31)
                        return &int_ctype;
                return left;
        }
        right = get_type(expr->right);
        if (!right)
                return NULL;

        if (left->type == SYM_PTR || left->type == SYM_ARRAY)
                return left;
        if (right->type == SYM_PTR || right->type == SYM_ARRAY)
                return right;

        if (type_positive_bits(left) < 31 && type_positive_bits(right) < 31)
                return &int_ctype;

        if (type_positive_bits(left) > type_positive_bits(right))
                return left;
        return right;
}

static struct symbol *get_type_symbol(struct expression *expr)
{
        if (!expr || expr->type != EXPR_SYMBOL || !expr->symbol)
                return NULL;

        return get_real_base_type(expr->symbol);
}

static struct symbol *get_member_symbol(struct symbol_list *symbol_list, struct
{
        struct symbol *tmp, *sub;

        FOR_EACH_PTR(symbol_list, tmp) {
                if (!tmp->ident) {
                        sub = get_real_base_type(tmp);
                        sub = get_member_symbol(sub->symbol_list, member);
                        if (sub)
                                return sub;
                        continue;
                }
                if (tmp->ident == member)
                        return tmp;
        } END_FOR_EACH_PTR(tmp);

        return NULL;
}

static struct symbol *get_symbol_from_deref(struct expression *expr)
{
        struct ident *member;
        struct symbol *sym;

        if (!expr || expr->type != EXPR_DEREF)
                return NULL;

        member = expr->member;
```

```
127         sym = get_type(expr->deref);
128         if (!sym) {
129                 // sm_msg("could not find struct type");
130                 return NULL;
131         }
132         if (sym->type == SYM_PTR)
133                 sym = get_real_base_type(sym);
134         sym = get_member_symbol(sym->symbol_list, member);
135         if (!sym)
136                 return NULL;
137         return get_real_base_type(sym);
138 }

140 static struct symbol *get_return_type(struct expression *expr)
141 {
142         struct symbol *tmp;

144         tmp = get_type(expr->fn);
145         if (!tmp)
146                 return NULL;
147         /* this is to handle __builtin_constant_p() */
148         if (tmp->type != SYM_FN)
149                 tmp = get_base_type(tmp);
150         return get_real_base_type(tmp);
151 }

153 static struct symbol *get_expr_stmt_type(struct statement *stmt)
154 {
155         if (stmt->type != STMT_COMPOUND)
156                 return NULL;
157         stmt = last_ptr_list((struct ptr_list *)stmt->stmts);
158         if (stmt->type == STMT_LABEL)
159                 stmt = stmt->label_statement;
160         if (stmt->type != STMT_EXPRESSION)
161                 return NULL;
162         return get_type(stmt->expression);
163 }

165 static struct symbol *get_select_type(struct expression *expr)
166 {
167         struct symbol *one, *two;

169         one = get_type(expr->cond_true);
170         two = get_type(expr->cond_false);
171         if (!one || !two)
172                 return NULL;
173         /*
174          * This is a hack.  If the types are not equiv then we
175          * really don't know the type.  But I think guessing is
176          *  probably Ok here.
177          */
178         if (type_positive_bits(one) > type_positive_bits(two))
179                 return one;
180         return two;
181 }

183 struct symbol *get_pointer_type(struct expression *expr)
184 {
185         struct symbol *sym;

187         sym = get_type(expr);
188         if (!sym)
189                 return NULL;
190         if (sym->type == SYM_NODE) {
191                 sym = get_real_base_type(sym);
192                 if (!sym)
```



```
193                         return NULL;
194         }
195         if (sym->type != SYM_PTR && sym->type != SYM_ARRAY)
196                 return NULL;
197         return get_real_base_type(sym);
198 }

200 static struct symbol *fake_pointer_sym(struct expression *expr)
201 {
202         struct symbol *sym;
203         struct symbol *base;

205         sym = alloc_symbol(expr->pos, SYM_PTR);
206         expr = expr->unop;
207         base = get_type(expr);
208         if (!base)
209                 return NULL;
210         sym->ctype.base_type = base;
211         return sym;
212 }

214 static struct symbol *get_type_helper(struct expression *expr)
215 {
216         struct symbol *ret;

218         expr = strip_parens(expr);
219         if (!expr)
220                 return NULL;

222         if (expr->ctype)
223                 return expr->ctype;

225         switch (expr->type) {
226         case EXPR_STRING:
227                 ret = &string_ctype;
228                 break;
229         case EXPR_SYMBOL:
230                 ret = get_type_symbol(expr);
231                 break;
232         case EXPR_DEREF:
233                 ret = get_symbol_from_deref(expr);
234                 break;
235         case EXPR_PREOP:
236         case EXPR_POSTOP:
237                 if (expr->op == '&')
238                         ret = fake_pointer_sym(expr);
239                 else if (expr->op == '*')
240                         ret = get_pointer_type(expr->unop);
241                 else
242                         ret = get_type(expr->unop);
243                 break;
244         case EXPR_ASSIGNMENT:
245                 ret = get_type(expr->left);
246                 break;
247         case EXPR_CAST:
248         case EXPR_FORCE_CAST:
249         case EXPR_IMPLIED_CAST:
250                 ret = get_real_base_type(expr->cast_type);
251                 break;
252         case EXPR_COMPARE:
253         case EXPR_BINOP:
254                 ret = get_binop_type(expr);
255                 break;
256         case EXPR_CALL:
257                 ret = get_return_type(expr);
258                 break;
```

```
259         case EXPR_STATEMENT:
260                 ret = get_expr_stmt_type(expr->statement);
261                 break;
262         case EXPR_CONDITIONAL:
263         case EXPR_SELECT:
264                 ret = get_select_type(expr);
265                 break;
266         case EXPR_SIZEOF:
267                 ret = &ulong_ctype;
268                 break;
269         case EXPR_LOGICAL:
270                 ret = &int_ctype;
271                 break;
272         default:
273                 return NULL;
274         }

276         if (ret && ret->type == SYM_TYPEOF)
277                 ret = get_type(ret->initializer);

279         expr->ctype = ret;
280         return ret;
281 }

283 static struct symbol *get_final_type_helper(struct expression *expr)
284 {
285         /*
286          * I'm not totally positive I understand types...
287          *
288          * So, when you're doing pointer math, and you do a subtraction, then
289          * the sval_binop() and whatever need to know the type of the pointer
290          * so they can figure out the alignment.  But the result is going to be
291          * and ssize_t.  So get_operation_type() gives you the pointer type
292          * and get_type() gives you ssize_t.
293          *
294          * Most of the time the operation type and the final type are the same
295          * but this just handles the few places where they are different.
296          *
297          */

299         expr = strip_parens(expr);
300         if (!expr)
301                 return NULL;

303         switch (expr->type) {
304         case EXPR_COMPARE:
305                 return &int_ctype;
306         case EXPR_BINOP: {
307                 struct symbol *left, *right;

309                 if (expr->op != '-')
310                         return NULL;

312                 left = get_type(expr->left);
313                 right = get_type(expr->right);
314                 if (type_is_ptr(left) || type_is_ptr(right))
315                         return ssize_t_ctype;
316                 }
317         }

319         return NULL;
320 }

322 struct symbol *get_type(struct expression *expr)
323 {
324         return get_type_helper(expr);
```



```
325 }

327 struct symbol *get_final_type(struct expression *expr)
328 {
329         struct symbol *ret;

331         ret = get_final_type_helper(expr);
332         if (ret)
333                 return ret;
334         return get_type_helper(expr);
335 }

337 struct symbol *get_promoted_type(struct symbol *left, struct symbol *right)
338 {
339         struct symbol *ret = &int_ctype;

341         if (type_positive_bits(left) > type_positive_bits(ret))
342                 ret = left;
343         if (type_positive_bits(right) > type_positive_bits(ret))
344                 ret = right;

346         if (type_is_ptr(left))
347                 ret = left;
348         if (type_is_ptr(right))
349                 ret = right;

351         return ret;
352 }

354 int type_signed(struct symbol *base_type)
355 {
356         if (!base_type)
357                 return 0;
358         if (base_type->ctype.modifiers & MOD_SIGNED)
359                 return 1;
360         return 0;
361 }

363 int expr_unsigned(struct expression *expr)
364 {
365         struct symbol *sym;

367         sym = get_type(expr);
368         if (!sym)
369                 return 0;
370         if (type_unsigned(sym))
371                 return 1;
372         return 0;
373 }

375 int expr_signed(struct expression *expr)
376 {
377         struct symbol *sym;

379         sym = get_type(expr);
380         if (!sym)
381                 return 0;
382         if (type_signed(sym))
383                 return 1;
384         return 0;
385 }

387 int returns_unsigned(struct symbol *sym)
388 {
389         if (!sym)
390                 return 0;
```

```
391         sym = get_base_type(sym);
392         if (!sym || sym->type != SYM_FN)
393                 return 0;
394         sym = get_base_type(sym);
395         return type_unsigned(sym);
396 }

398 int is_pointer(struct expression *expr)
399 {
400         struct symbol *sym;

402         sym = get_type(expr);
403         if (!sym)
404                 return 0;
405         if (sym == &string_ctype)
406                 return 0;
407         if (sym->type == SYM_PTR)
408                 return 1;
409         return 0;
410 }

412 int returns_pointer(struct symbol *sym)
413 {
414         if (!sym)
415                 return 0;
416         sym = get_base_type(sym);
417         if (!sym || sym->type != SYM_FN)
418                 return 0;
419         sym = get_base_type(sym);
420         if (sym->type == SYM_PTR)
421                 return 1;
422         return 0;
423 }

425 sval_t sval_type_max(struct symbol *base_type)
426 {
427         sval_t ret;

429         if (!base_type || !type_bits(base_type))
430                 base_type = &llong_ctype;
431         ret.type = base_type;

433         ret.value = (~0ULL) >> (64 - type_positive_bits(base_type));
434         return ret;
435 }

437 sval_t sval_type_min(struct symbol *base_type)
438 {
439         sval_t ret;

441         if (!base_type || !type_bits(base_type))
442                 base_type = &llong_ctype;
443         ret.type = base_type;

445         if (type_unsigned(base_type)) {
446                 ret.value = 0;
447                 return ret;
448         }

450         ret.value = (~0ULL) << type_positive_bits(base_type);

452         return ret;
453 }

455 int nr_bits(struct expression *expr)
456 {
```



```
457         struct symbol *type;

459         type = get_type(expr);
460         if (!type)
461                 return 0;
462         return type_bits(type);
463 }

465 int is_void_pointer(struct expression *expr)
466 {
467         struct symbol *type;

469         type = get_type(expr);
470         if (!type || type->type != SYM_PTR)
471                 return 0;
472         type = get_real_base_type(type);
473         if (type == &void_ctype)
474                 return 1;
475         return 0;
476 }

478 int is_char_pointer(struct expression *expr)
479 {
480         struct symbol *type;

482         type = get_type(expr);
483         if (!type || type->type != SYM_PTR)
484                 return 0;
485         type = get_real_base_type(type);
486         if (type == &char_ctype)
487                 return 1;
488         return 0;
489 }

491 int is_string(struct expression *expr)
492 {
493         expr = strip_expr(expr);
494         if (!expr || expr->type != EXPR_STRING)
495                 return 0;
496         if (expr->string)
497                 return 1;
498         return 0;
499 }

501 int is_static(struct expression *expr)
502 {
503         char *name;
504         struct symbol *sym;
505         int ret = 0;

507         name = expr_to_str_sym(expr, &sym);
508         if (!name || !sym)
509                 goto free;

511         if (sym->ctype.modifiers & MOD_STATIC)
512                 ret = 1;
513 free:
514         free_string(name);
515         return ret;
516 }

518 int is_local_variable(struct expression *expr)
519 {
520         struct symbol *sym;
521         char *name;
```

```
523         name = expr_to_var_sym(expr, &sym);
524         free_string(name);
525         if (!sym || !sym->scope || !sym->scope->token || !cur_func_sym)
526                 return 0;
527         if (cmp_pos(sym->scope->token->pos, cur_func_sym->pos) < 0)
528                 return 0;
529         if (is_static(expr))
530                 return 0;
531         return 1;
532 }

534 int types_equiv(struct symbol *one, struct symbol *two)
535 {
536         if (!one && !two)
537                 return 1;
538         if (!one || !two)
539                 return 0;
540         if (one->type != two->type)
541                 return 0;
542         if (one->type == SYM_PTR)
543                 return types_equiv(get_real_base_type(one), get_real_base_type(t
544         if (type_positive_bits(one) != type_positive_bits(two))
545                 return 0;
546         return 1;
547 }

549 int fn_static(void)
550 {
551         return !!(cur_func_sym->ctype.modifiers & MOD_STATIC);
552 }

554 const char *global_static(void)
555 {
556         if (cur_func_sym->ctype.modifiers & MOD_STATIC)
557                 return "static";
558         else
559                 return "global";
560 }

562 struct symbol *cur_func_return_type(void)
563 {
564         struct symbol *sym;

566         sym = get_real_base_type(cur_func_sym);
567         if (!sym || sym->type != SYM_FN)
568                 return NULL;
569         sym = get_real_base_type(sym);
570         return sym;
571 }

573 struct symbol *get_arg_type(struct expression *fn, int arg)
574 {
575         struct symbol *fn_type;
576         struct symbol *tmp;
577         struct symbol *arg_type;
578         int i;

580         fn_type = get_type(fn);
581         if (!fn_type)
582                 return NULL;
583         if (fn_type->type == SYM_PTR)
584                 fn_type = get_real_base_type(fn_type);
585         if (fn_type->type != SYM_FN)
586                 return NULL;

588         i = 0;
```



```
589         FOR_EACH_PTR(fn_type->arguments, tmp) {
590                 arg_type = get_real_base_type(tmp);
591                 if (i == arg) {
592                         return arg_type;
593                 }
594                 i++;
595         } END_FOR_EACH_PTR(tmp);

597         return NULL;
598 }

600 static struct symbol *get_member_from_string(struct symbol_list *symbol_list, co
601 {
602         struct symbol *tmp, *sub;
603         int chunk_len;

605         if (strncmp(name, ".", 1) == 0)
606                 name += 1;
607         if (strncmp(name, "->", 2) == 0)
608                 name += 2;

610         FOR_EACH_PTR(symbol_list, tmp) {
611                 if (!tmp->ident) {
612                         sub = get_real_base_type(tmp);
613                         sub = get_member_from_string(sub->symbol_list, name);
614                         if (sub)
615                                 return sub;
616                         continue;
617                 }

619                 if (strcmp(tmp->ident->name, name) == 0)
620                         return tmp;

622                 chunk_len = strlen(tmp->ident->name);
623                 if (strncmp(tmp->ident->name, name, chunk_len) == 0 &&
624                     (name[chunk_len] == '.' || name[chunk_len] == '-')) {
625                         sub = get_real_base_type(tmp);
626                         return get_member_from_string(sub->symbol_list, name + c
627                 }

629         } END_FOR_EACH_PTR(tmp);

631         return NULL;
632 }

634 struct symbol *get_member_type_from_key(struct expression *expr, const char *key
635 {
636         struct symbol *sym;

638         if (strcmp(key, "$") == 0)
639                 return get_type(expr);

641         if (strcmp(key, "*$") == 0) {
642                 sym = get_type(expr);
643                 if (!sym || sym->type != SYM_PTR)
644                         return NULL;
645                 return get_real_base_type(sym);
646         }

648         sym = get_type(expr);
649         if (!sym)
650                 return NULL;
651         if (sym->type == SYM_PTR)
652                 sym = get_real_base_type(sym);

654         key = key + 1;
```

```
655         sym = get_member_from_string(sym->symbol_list, key);
656         if (!sym)
657                 return NULL;
658         return get_real_base_type(sym);
659 }

661 struct symbol *get_arg_type_from_key(struct expression *fn, int param, struct ex
662 {
663         struct symbol *type;

665         if (!key)
666                 return NULL;
667         if (strcmp(key, "$") == 0)
668                 return get_arg_type(fn, param);
669         if (strcmp(key, "*$") == 0) {
670                 type = get_arg_type(fn, param);
671                 if (!type || type->type != SYM_PTR)
672                         return NULL;
673                 return get_real_base_type(type);
674         }
675         return get_member_type_from_key(arg, key);
676 }

678 int is_struct(struct expression *expr)
679 {
680         struct symbol *type;

682         type = get_type(expr);
683         if (type && type->type == SYM_STRUCT)
684                 return 1;
685         return 0;
686 }

688 static struct {
689         struct symbol *sym;
690         const char *name;
691 } base_types[] = {
692         {&bool_ctype, "bool"},
693         {&void_ctype, "void"},
694         {&type_ctype, "type"},
695         {&char_ctype, "char"},
696         {&schar_ctype, "schar"},
697         {&uchar_ctype, "uchar"},
698         {&short_ctype, "short"},
699         {&sshort_ctype, "sshort"},
700         {&ushort_ctype, "ushort"},
701         {&int_ctype, "int"},
702         {&sint_ctype, "sint"},
703         {&uint_ctype, "uint"},
704         {&long_ctype, "long"},
705         {&slong_ctype, "slong"},
706         {&ulong_ctype, "ulong"},
707         {&llong_ctype, "llong"},
708         {&sllong_ctype, "sllong"},
709         {&ullong_ctype, "ullong"},
710         {&lllong_ctype, "lllong"},
711         {&slllong_ctype, "slllong"},
712         {&ulllong_ctype, "ulllong"},
713         {&float_ctype, "float"},
714         {&double_ctype, "double"},
715         {&ldouble_ctype, "ldouble"},
716         {&string_ctype, "string"},
717         {&ptr_ctype, "ptr"},
718         {&lazy_ptr_ctype, "lazy_ptr"},
719         {&incomplete_ctype, "incomplete"},
720         {&label_ctype, "label"},
```



```
721         {&bad_ctype, "bad"},
722         {&null_ctype, "null"},
723 };

725 static const char *base_type_str(struct symbol *sym)
726 {
727         int i;

729         for (i = 0; i < ARRAY_SIZE(base_types); i++) {
730                 if (sym == base_types[i].sym)
731                         return base_types[i].name;
732         }
733         return "<unknown>";
734 }

736 static int type_str_helper(char *buf, int size, struct symbol *type)
737 {
738         int n;

740         if (!type)
741                 return snprintf(buf, size, "<unknown>");

743         if (type->type == SYM_BASETYPE) {
744                 return snprintf(buf, size, base_type_str(type));
745         } else if (type->type == SYM_PTR) {
746                 type = get_real_base_type(type);
747                 n = type_str_helper(buf, size, type);
748                 if (n > size)
749                         return n;
750                 return n + snprintf(buf + n, size - n, "*");
751         } else if (type->type == SYM_ARRAY) {
752                 type = get_real_base_type(type);
753                 n = type_str_helper(buf, size, type);
754                 if (n > size)
755                         return n;
756                 return n + snprintf(buf + n, size - n, "[]");
757         } else if (type->type == SYM_STRUCT) {
758                 return snprintf(buf, size, "struct %s", type->ident ? type->iden
759         } else if (type->type == SYM_UNION) {
760                 if (type->ident)
761                         return snprintf(buf, size, "union %s", type->ident->name
762                 else
763                         return snprintf(buf, size, "anonymous union");
764         } else if (type->type == SYM_FN) {
765                 struct symbol *arg, *return_type, *arg_type;
766                 int i;

768                 return_type = get_real_base_type(type);
769                 n = type_str_helper(buf, size, return_type);
770                 if (n > size)
771                         return n;
772                 n += snprintf(buf + n, size - n, "(*)(");
773                 if (n > size)
774                         return n;

776                 i = 0;
777                 FOR_EACH_PTR(type->arguments, arg) {
778                         if (i++)
779                                 n += snprintf(buf + n, size - n, ", ");
780                         if (n > size)
781                                 return n;
782                         arg_type = get_real_base_type(arg);
783                         n += type_str_helper(buf + n, size - n, arg_type);
784                         if (n > size)
785                                 return n;
786                 } END_FOR_EACH_PTR(arg);
```

```
788                 return n + snprintf(buf + n, size - n, ")");
789         } else if (type->type == SYM_NODE) {
790                 n = snprintf(buf, size, "node {");
791                 if (n > size)
792                         return n;
793                 type = get_real_base_type(type);
794                 n += type_str_helper(buf + n, size - n, type);
795                 if (n > size)
796                         return n;
797                 return n + snprintf(buf + n, size - n, "}");
798         } else {
799                 return snprintf(buf, size, "<type %d>", type->type);
800         }
801 }

803 char *type_to_str(struct symbol *type)
804 {
805         static char buf[256];

807         buf[0] = '\0';
808         type_str_helper(buf, sizeof(buf), type);
809         return buf;
810 }
```

```
**********************************************************
    2026 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_type_links.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The plan here is to save all the possible values store to a given struct
 * member.
 *
 * We will load all the values in to the function_type_val table first then
 * run a script on that and load all the resulting values into the type_val
 * table.
 *
 * So in this file we want to take the union of everything assigned to the
 * struct member and insert it into the function_type_val at the end.
 *
 * You would think that we could use smatch_modification_hooks.c or
 * extra_modification_hook() here to get the information here but in the end we
 * need to code everything again a third time.
 *
 */

/*
 * Remember links like:
 *
 * foo->void_ptr = some_struct.
 *
 * If we get a some_struct pointer from foo->void_ptr then assume it's the same
 * stuff.
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

static void match_assign(struct expression *expr)
{
        struct symbol *type;

        if (!is_void_pointer(expr->left))
                return;

        type = get_type(expr->right);
        if (!type || type->type != SYM_PTR)
                return;
        type = get_real_base_type(type);
```



```
        if (!type || type->type != SYM_STRUCT)
                return;

        sql_insert_data_info(expr->left, TYPE_LINK, type_to_str(type));
}

void register_type_links(int id)
{
        if (!option_info)
                return;
        my_id = id;

        add_hook(&match_assign, ASSIGNMENT_HOOK);
}
```

```
**********************************************************
   14202 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_type_val.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The plan here is to save all the possible values store to a given struct
 * member.
 *
 * We will load all the values in to the function_type_val table first then
 * run a script on that and load all the resulting values into the type_val
 * table.
 *
 * So in this file we want to take the union of everything assigned to the
 * struct member and insert it into the function_type_val at the end.
 *
 * You would think that we could use smatch_modification_hooks.c or
 * extra_modification_hook() here to get the information here but in the end we
 * need to code everything again a third time.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;

struct stree_stack *fn_type_val_stack;
struct stree *fn_type_val;
struct stree *global_type_val;

static int get_vals(void *_db_vals, int argc, char **argv, char **azColName)
{
        char **db_vals = _db_vals;

        *db_vals = alloc_string(argv[0]);
        return 0;
}

static void match_inline_start(struct expression *expr)
{
        push_stree(&fn_type_val_stack, fn_type_val);
        fn_type_val = NULL;
}

static void match_inline_end(struct expression *expr)
{
```



```
        free_stree(&fn_type_val);
        fn_type_val = pop_stree(&fn_type_val_stack);
}

struct expr_rl {
        struct expression *expr;
        struct range_list *rl;
};
static struct expr_rl cached_results[10];
static int res_idx;

static int get_cached(struct expression *expr, struct range_list **rl, int *ret)
{
        int i;

        *ret = 0;

        for (i = 0; i < ARRAY_SIZE(cached_results); i++) {
                if (expr == cached_results[i].expr) {
                        if (cached_results[i].rl) {
                                *rl = clone_rl(cached_results[i].rl);
                                *ret = 1;
                        }
                        return 1;
                }
        }

        return 0;
}

int get_db_type_rl(struct expression *expr, struct range_list **rl)
{
        char *db_vals = NULL;
        char *member;
        struct range_list *tmp;
        struct symbol *type;
        int ret;

        if (get_cached(expr, rl, &ret))
                return ret;

        member = get_member_name(expr);
        if (!member)
                return 0;

        res_idx = (res_idx + 1) % ARRAY_SIZE(cached_results);
        cached_results[res_idx].expr = expr;
        cached_results[res_idx].rl = NULL;

        run_sql(get_vals, &db_vals,
                "select value from type_value where type = '%s';", member);
        free_string(member);
        if (!db_vals)
                return 0;
        type = get_type(expr);
        str_to_rl(type, db_vals, &tmp);
        free_string(db_vals);
        if (is_whole_rl(tmp))
                return 0;

        *rl = tmp;
        cached_results[res_idx].rl = clone_rl(tmp);

        return 1;
}
```

```
127 static void add_type_val(char *member, struct range_list *rl)
128 {
129         struct smatch_state *old, *add, *new;

131         member = alloc_string(member);
132         old = get_state_stree(fn_type_val, my_id, member, NULL);
133         add = alloc_estate_rl(rl);
134         if (old)
135                 new = merge_estates(old, add);
136         else
137                 new = add;
138         set_state_stree(&fn_type_val, my_id, member, NULL, new);
139 }

141 static void add_fake_type_val(char *member, struct range_list *rl, int ignore)
142 {
143         struct smatch_state *old, *add, *new;

145         member = alloc_string(member);
146         old = get_state_stree(fn_type_val, my_id, member, NULL);
147         if (old && strcmp(old->name, "min-max") == 0)
148                 return;
149         if (ignore && old && strcmp(old->name, "ignore") == 0)
150                 return;
151         add = alloc_estate_rl(rl);
152         if (old) {
153                 new = merge_estates(old, add);
154         } else {
155                 new = add;
156                 if (ignore)
157                         new->name = alloc_string("ignore");
158                 else
159                         new->name = alloc_string("min-max");
160         }
161         set_state_stree(&fn_type_val, my_id, member, NULL, new);
162 }

164 static void add_global_type_val(char *member, struct range_list *rl)
165 {
166         struct smatch_state *old, *add, *new;

168         member = alloc_string(member);
169         old = get_state_stree(global_type_val, my_id, member, NULL);
170         add = alloc_estate_rl(rl);
171         if (old)
172                 new = merge_estates(old, add);
173         else
174                 new = add;
175         new = clone_estate_perm(new);
176         set_state_stree_perm(&global_type_val, my_id, member, NULL, new);
177 }

179 static int has_link_cb(void *has_link, int argc, char **argv, char **azColName)
180 {
181         *(int *)has_link = 1;
182         return 0;
183 }

185 static int is_ignored_fake_assignment(void)
186 {
187         struct expression *expr;
188         struct symbol *type;
189         char *member_name;
190         int has_link = 0;

192         expr = get_faked_expression();
```



```
193         if (!expr || expr->type != EXPR_ASSIGNMENT)
194                 return 0;
195         if (!is_void_pointer(expr->right))
196                 return 0;
197         member_name = get_member_name(expr->right);
198         if (!member_name)
199                 return 0;

201         type = get_type(expr->left);
202         if (!type || type->type != SYM_PTR)
203                 return 0;
204         type = get_real_base_type(type);
205         if (!type || type->type != SYM_STRUCT)
206                 return 0;

208         run_sql(has_link_cb, &has_link,
209                 "select * from data_info where type = %d and data = '%s' and val
210                 TYPE_LINK, member_name, type_to_str(type));
211         return has_link;
212 }

214 static int is_container_of(void)
215 {
216         /* We already check the macro name in is_ignored_macro() */
217         struct expression *expr;
218         int offset;

220         expr = get_faked_expression();
221         if (!expr || expr->type != EXPR_ASSIGNMENT)
222                 return 0;

224         offset = get_offset_from_container_of(expr->right);
225         if (offset < 0)
226                 return 0;
227         return 1;
228 }

230 static int is_ignored_macro(void)
231 {
232         struct expression *expr;
233         char *name;

235         expr = get_faked_expression();
236         if (!expr || expr->type != EXPR_ASSIGNMENT)
237                 return 0;
238         name = get_macro_name(expr->right->pos);
239         if (!name)
240                 return 0;
241         if (strcmp(name, "container_of") == 0)
242                 return 1;
243         if (strcmp(name, "rb_entry") == 0)
244                 return 1;
245         if (strcmp(name, "list_entry") == 0)
246                 return 1;
247         if (strcmp(name, "list_first_entry") == 0)
248                 return 1;
249         if (strcmp(name, "hlist_entry") == 0)
250                 return 1;
251         if (strstr(name, "for_each"))
252                 return 1;
253         return 0;
254 }

256 static int is_ignored_function(void)
257 {
258         struct expression *expr;
```

```
260         expr = get_faked_expression();
261         if (!expr || expr->type != EXPR_ASSIGNMENT)
262                 return 0;
263         expr = strip_expr(expr->right);
264         if (!expr || expr->type != EXPR_CALL || expr->fn->type != EXPR_SYMBOL)
265                 return 0;

267         if (sym_name_is("kmalloc", expr->fn))
268                 return 1;
269         if (sym_name_is("netdev_priv", expr->fn))
270                 return 1;
271         if (sym_name_is("dev_get_drvdata", expr->fn))
272                 return 1;

274         return 0;
275 }

277 static int is_uncasted_pointer_assign(void)
278 {
279         struct expression *expr;
280         struct symbol *left_type, *right_type;

282         expr = get_faked_expression();
283         if (!expr)
284                 return 0;
285         if (expr->type == EXPR_PREOP || expr->type == EXPR_POSTOP) {
286                 if (expr->op == SPECIAL_INCREMENT || expr->op == SPECIAL_DECREME
287                         return 1;
288         }
289         if (expr->type != EXPR_ASSIGNMENT)
290                 return 0;
291         left_type = get_type(expr->left);
292         right_type = get_type(expr->right);

294         if (!left_type || !right_type)
295                 return 0;

297         if (left_type->type != SYM_PTR &&
298             left_type->type != SYM_ARRAY)
299                 return 0;
300         if (right_type->type != SYM_PTR &&
301             right_type->type != SYM_ARRAY)
302                 return 0;
303         left_type = get_real_base_type(left_type);
304         right_type = get_real_base_type(right_type);

306         if (left_type == right_type)
307                 return 1;
308         return 0;
309 }

311 static int set_param_type(void *_type_str, int argc, char **argv, char **azColNa
312 {
313         char **type_str = _type_str;
314         static char type_buf[128];

316         if (*type_str) {
317                 if (strcmp(*type_str, argv[0]) == 0)
318                         return 0;
319                 strncpy(type_buf, "unknown", sizeof(type_buf));
320                 return 0;
321         }
322         strncpy(type_buf, argv[0], sizeof(type_buf));
323         *type_str = type_buf;
```



```
325         return 0;
326 }

328 static char *db_get_parameter_type(int param)
329 {
330         char *ret = NULL;

332         if (!cur_func_sym)
333                 return NULL;

335         run_sql(set_param_type, &ret,
336                 "select value from fn_data_link where "
337                 "file = '%s' and function = '%s' and static = %d and type = %d a
338                 (cur_func_sym->ctype.modifiers & MOD_STATIC) ? get_base_file() :
339                 cur_func_sym->ident->name,
340                 !!(cur_func_sym->ctype.modifiers & MOD_STATIC),
341                 PASSES_TYPE, param);

343         return ret;
344 }

346 static int is_uncasted_fn_param_from_db(void)
347 {
348         struct expression *expr, *right;
349         struct symbol *left_type;
350         char left_type_name[128];
351         int param;
352         char *right_type_name;
353         static struct expression *prev_expr;
354         static int prev_ans;

356         expr = get_faked_expression();

358         if (expr == prev_expr)
359                 return prev_ans;
360         prev_expr = expr;
361         prev_ans = 0;

363         if (!expr || expr->type != EXPR_ASSIGNMENT)
364                 return 0;
365         left_type = get_type(expr->left);
366         if (!left_type || left_type->type != SYM_PTR)
367                 return 0;
368         left_type = get_real_base_type(left_type);
369         if (!left_type || left_type->type != SYM_STRUCT)
370                 return 0;
371         snprintf(left_type_name, sizeof(left_type_name), "%s", type_to_str(left_

373         right = strip_expr(expr->right);
374         param = get_param_num(right);
375         if (param < 0)
376                 return 0;
377         right_type_name = db_get_parameter_type(param);
378         if (!right_type_name)
379                 return 0;

381         if (strcmp(right_type_name, left_type_name) == 0) {
382                 prev_ans = 1;
383                 return 1;
384         }

386         return 0;
387 }

389 static void match_assign_value(struct expression *expr)
390 {
```

```
391         char *member, *right_member;
392         struct range_list *rl;
393         struct symbol *type;

395         if (!cur_func_sym)
396                 return;

398         type = get_type(expr->left);
399         if (type && type->type == SYM_STRUCT)
400                 return;

402         member = get_member_name(expr->left);
403         if (!member)
404                 return;

406         /* if we're saying foo->mtu = bar->mtu then that doesn't add information
407         right_member = get_member_name(expr->right);
408         if (right_member && strcmp(right_member, member) == 0)
409                 goto free;

411         if (is_fake_call(expr->right)) {
412                 if (is_ignored_macro())
413                         goto free;
414                 if (is_ignored_function())
415                         goto free;
416                 if (is_uncasted_pointer_assign())
417                         goto free;
418                 if (is_uncasted_fn_param_from_db())
419                         goto free;
420                 if (is_container_of())
421                         goto free;
422                 add_fake_type_val(member, alloc_whole_rl(get_type(expr->left)),
423                 goto free;
424         }

426         if (expr->op == '=') {
427                 get_absolute_rl(expr->right, &rl);
428                 rl = cast_rl(type, rl);
429         } else {
430                 /*
431                  * This is a bit cheating.  We order it so this will already be
432                  * by smatch_extra.c and we just look up the value.
433                  */
434                 get_absolute_rl(expr->left, &rl);
435         }
436         add_type_val(member, rl);
437 free:
438         free_string(right_member);
439         free_string(member);
440 }

442 /*
443  * If we too:  int *p = &my_struct->member then abandon all hope of tracking
444  * my_struct->member.
445  */
446 static void match_assign_pointer(struct expression *expr)
447 {
448         struct expression *right;
449         char *member;
450         struct range_list *rl;
451         struct symbol *type;

453         right = strip_expr(expr->right);
454         if (right->type != EXPR_PREOP || right->op != '&')
455                 return;
456         right = strip_expr(right->unop);
```



```
458         member = get_member_name(right);
459         if (!member)
460                 return;
461         type = get_type(right);
462         rl = alloc_whole_rl(type);
463         add_type_val(member, rl);
464         free_string(member);
465 }

467 static void match_global_assign(struct expression *expr)
468 {
469         char *member;
470         struct range_list *rl;
471         struct symbol *type;

473         type = get_type(expr->left);
474         if (type && (type->type == SYM_ARRAY || type->type == SYM_STRUCT))
475                 return;
476         member = get_member_name(expr->left);
477         if (!member)
478                 return;
479         get_absolute_rl(expr->right, &rl);
480         rl = cast_rl(type, rl);
481         add_global_type_val(member, rl);
482         free_string(member);
483 }

485 static void unop_expr(struct expression *expr)
486 {
487         struct range_list *rl;
488         char *member;

490         if (expr->op != SPECIAL_DECREMENT && expr->op != SPECIAL_INCREMENT)
491                 return;

493         expr = strip_expr(expr->unop);
494         member = get_member_name(expr);
495         if (!member)
496                 return;
497         rl = alloc_whole_rl(get_type(expr));
498         add_type_val(member, rl);
499         free_string(member);
500 }

502 static void asm_expr(struct statement *stmt)
503 {
504         struct expression *expr;
505         struct range_list *rl;
506         char *member;
507         int state = 0;

509         FOR_EACH_PTR(stmt->asm_outputs, expr) {
510                 switch (state) {
511                 case 0: /* identifier */
512                 case 1: /* constraint */
513                         state++;
514                         continue;
515                 case 2: /* expression */
516                         state = 0;
517                         member = get_member_name(expr);
518                         if (!member)
519                                 continue;
520                         rl = alloc_whole_rl(get_type(expr));
521                         add_type_val(member, rl);
522                         free_string(member);
```

```
523                         continue;
524                 }
525         } END_FOR_EACH_PTR(expr);
526 }

528 static void db_param_add(struct expression *expr, int param, char *key, char *va
529 {
530         struct expression *arg;
531         struct symbol *type;
532         struct range_list *rl;
533         char *member;

535         if (strcmp(key, "*$") != 0)
536                 return;

538         while (expr->type == EXPR_ASSIGNMENT)
539                 expr = strip_expr(expr->right);
540         if (expr->type != EXPR_CALL)
541                 return;

543         arg = get_argument_from_call_expr(expr->args, param);
544         arg = strip_expr(arg);
545         if (!arg)
546                 return;
547         type = get_member_type_from_key(arg, key);
548         if (arg->type != EXPR_PREOP || arg->op != '&')
549                 return;
550         arg = strip_expr(arg->unop);

552         member = get_member_name(arg);
553         if (!member)
554                 return;
555         call_results_to_rl(expr, type, value, &rl);
556         add_type_val(member, rl);
557         free_string(member);
558 }

560 static void match_end_func_info(struct symbol *sym)
561 {
562         struct sm_state *sm;

564         FOR_EACH_SM(fn_type_val, sm) {
565                 sql_insert_function_type_value(sm->name, sm->state->name);
566         } END_FOR_EACH_SM(sm);
567 }

569 static void clear_cache(struct symbol *sym)
570 {
571         memset(cached_results, 0, sizeof(cached_results));
572 }

574 static void match_after_func(struct symbol *sym)
575 {
576         free_stree(&fn_type_val);
577 }

579 static void match_end_file(struct symbol_list *sym_list)
580 {
581         struct sm_state *sm;

583         FOR_EACH_SM(global_type_val, sm) {
584                 sql_insert_function_type_value(sm->name, sm->state->name);
585         } END_FOR_EACH_SM(sm);
586 }

588 void register_type_val(int id)
```



```
589 {
590         my_id = id;
591         add_hook(&clear_cache, AFTER_FUNC_HOOK);

593         if (!option_info)
594                 return;

596         add_hook(&match_assign_value, ASSIGNMENT_HOOK_AFTER);
597         add_hook(&match_assign_pointer, ASSIGNMENT_HOOK);
598         add_hook(&unop_expr, OP_HOOK);
599         add_hook(&asm_expr, ASM_HOOK);
600         select_return_states_hook(PARAM_ADD, &db_param_add);
601         select_return_states_hook(PARAM_SET, &db_param_add);


604         add_hook(&match_inline_start, INLINE_FN_START);
605         add_hook(&match_inline_end, INLINE_FN_END);

607         add_hook(&match_end_func_info, END_FUNC_HOOK);
608         add_hook(&match_after_func, AFTER_FUNC_HOOK);

610         add_hook(&match_global_assign, GLOBAL_ASSIGNMENT_HOOK);
611         add_hook(&match_end_file, END_FILE_HOOK);
612 }
```

```
**********************************************************
    1971 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_unknown_value.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * The situation here is that we often need to fake an assignment but we don't
 * know anything about the right hand side of the assignment.  We use a fake
 * function call of &llong_ctype.  The reason for using a function call instead
 * of a value is so we don't start storing the equivalence.
 *
 */

#include "smatch.h"

struct ident fake_assign = {
        .len = sizeof("fake assign"),
        .name = "fake assign",
};

static struct symbol fake_fn_symbol = {
        .type = SYM_FN,
        .ident = &fake_assign,
};

static struct symbol fake_node_symbol = {
        .type = SYM_NODE,
        .ident = &fake_assign,
};

static struct expression fake_fn_expr = {
        .type = EXPR_SYMBOL,
        .ctype = &llong_ctype,
};

static struct expression fake_call = {
        .type = EXPR_CALL,
        .ctype = &llong_ctype,
};

static void __attribute__((constructor)) initialize_local_variables(void)
{
        fake_fn_symbol.ctype.base_type = &llong_ctype;
        fake_node_symbol.ctype.base_type = &fake_fn_symbol;
        fake_fn_expr.symbol = &fake_node_symbol;
        fake_fn_expr.symbol_name = &fake_assign;
        fake_call.fn = &fake_fn_expr;
}
```



```
struct expression *unknown_value_expression(struct expression *expr)
{
        fake_fn_expr.parent = 0;
        fake_call.parent = 0;
        return &fake_call;
}

int is_fake_call(struct expression *expr)
{
        return expr == &fake_call;
}
```

```
**********************************************************
    5808 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_untracked_param.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2014 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

/*
 * Sometimes we aren't able to track a variable through a function call.  This
 * usually happens because a function changes too many variables so we give up.
 * Another reason this happens is because we call a function pointer and there
 * are too many functions which implement that function pointer so we give up.
 * Also maybe we don't have the database enabled.
 *
 * The goal here is to make a call back so what if we call:
 *
 *      frob(&foo);
 *
 * but we're not able to say what happens to "foo", then let's assume that we
 * don't know anything about "foo" if it's an untracked call.
 *
 */

#include "smatch.h"
#include "smatch_slist.h"
#include "smatch_extra.h"

static int my_id;
static int tracked;

STATE(untracked);

typedef void (untracked_hook)(struct expression *call, int param);
DECLARE_PTR_LIST(untracked_hook_list, untracked_hook *);
static struct untracked_hook_list *untracked_hooks;

struct int_stack *tracked_stack;

void add_untracked_param_hook(void (func)(struct expression *call, int param))
{
        untracked_hook **p = malloc(sizeof(untracked_hook *));
        *p = func;
        add_ptr_list(&untracked_hooks, p);
}

static void call_untracked_callbacks(struct expression *expr, int param)
{
        untracked_hook **fn;

        FOR_EACH_PTR(untracked_hooks, fn) {
```



```
                (*fn)(expr, param);
        } END_FOR_EACH_PTR(fn);
}

static void assume_tracked(struct expression *call_expr, int param, char *key, c
{
        tracked = 1;
}

void mark_untracked(struct expression *expr, int param, const char *key, const c
{
        char *name;
        struct symbol *sym;

        while (expr->type == EXPR_ASSIGNMENT)
                expr = strip_expr(expr->right);
        if (expr->type != EXPR_CALL)
                return;

        name = return_state_to_var_sym(expr, param, key, &sym);
        if (!name || !sym)
                goto free;

        call_untracked_callbacks(expr, param);
        set_state(my_id, name, sym, &untracked);
free:
        free_string(name);
}

static int lost_in_va_args(struct expression *expr)
{
        struct symbol *fn;
        char *name;
        int is_lost;

        fn = get_type(expr->fn);
        if (!fn || !fn->variadic)
                return 0;

        is_lost = 1;
        name = expr_to_var(expr->fn);
        if (name && strstr(name, "print"))
                is_lost = 0;
        free_string(name);

        return is_lost;
}

static void match_after_call(struct expression *expr)
{
        struct expression *arg;
        struct symbol *type;
        int i;

        if (lost_in_va_args(expr))
                tracked = 0;

        if (tracked) {
                tracked = 0;
                return;
        }

        i = -1;
        FOR_EACH_PTR(expr->args, arg) {
                i++;
```

```
127                 type = get_type(arg);
128                 if (!type || type->type != SYM_PTR)
129                         continue;
130
131                 call_untracked_callbacks(expr, i);
132                 set_state_expr(my_id, arg, &untracked);
133         } END_FOR_EACH_PTR(arg);
134 }
135
136 void mark_all_params_untracked(int return_id, char *return_ranges, struct expres
137 {
138         struct symbol *arg;
139         int param;
140
141         param = -1;
142         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
143                 param++;
144
145                 if (!arg->ident)
146                         continue;
147                 sql_insert_return_states(return_id, return_ranges,
148                                          UNTRACKED_PARAM, param, "$", "");
149         } END_FOR_EACH_PTR(arg);
150 }
151
152 static void print_untracked_params(int return_id, char *return_ranges, struct ex
153 {
154         struct symbol *arg;
155         int param;
156
157         param = -1;
158         FOR_EACH_PTR(cur_func_sym->ctype.base_type->arguments, arg) {
159                 param++;
160
161                 if (!arg->ident)
162                         continue;
163                 if (!get_state(my_id, arg->ident->name, arg) &&
164                     !__bail_on_rest_of_function)  /* hairy functions are untrack
165                         continue;
166
167                 sql_insert_return_states(return_id, return_ranges,
168                                          UNTRACKED_PARAM, param, "$", "");
169         } END_FOR_EACH_PTR(arg);
170 }
171
172 static void match_param_assign(struct expression *expr)
173 {
174         struct expression *right;
175         struct symbol *type;
176         int param;
177
178         if (__in_fake_assign)
179                 return;
180
181         right = strip_expr(expr->right);
182         type = get_type(right);
183         if (!type || type->type != SYM_PTR)
184                 return;
185
186         param = get_param_num(right);
187         if (param < 0)
188                 return;
189
190         set_state_expr(my_id, right, &untracked);
191 }
```



```
194 static void match_param_assign_in_asm(struct statement *stmt)
195 {
196
197         struct expression *expr;
198         struct symbol *type;
199         int state = 0;
200         int param;
201
202         FOR_EACH_PTR(stmt->asm_inputs, expr) {
203                 switch (state) {
204                 case 0: /* identifier */
205                 case 1: /* constraint */
206                         state++;
207                         continue;
208                 case 2: /* expression */
209                         state = 0;
210
211                         expr = strip_expr(expr);
212                         type = get_type(expr);
213                         if (!type || type->type != SYM_PTR)
214                                 continue;
215                         param = get_param_num(expr);
216                         if (param < 0)
217                                 continue;
218                         set_state_expr(my_id, expr, &untracked);
219                         continue;
220                 }
221         } END_FOR_EACH_PTR(expr);
222 }
223
224 static void match_inline_start(struct expression *expr)
225 {
226         push_int(&tracked_stack, tracked);
227 }
228
229 static void match_inline_end(struct expression *expr)
230 {
231         tracked = pop_int(&tracked_stack);
232 }
233
234 void register_untracked_param(int id)
235 {
236         my_id = id;
237
238         select_return_states_hook(INTERNAL, &assume_tracked);
239         select_return_states_hook(UNTRACKED_PARAM, &mark_untracked);
240         add_hook(&match_after_call, FUNCTION_CALL_HOOK_AFTER_DB);
241
242         add_split_return_callback(&print_untracked_params);
243
244         add_hook(&match_param_assign, ASSIGNMENT_HOOK);
245         add_hook(&match_param_assign_in_asm, ASM_HOOK);
246
247         add_hook(&match_inline_start, INLINE_FN_START);
248         add_hook(&match_inline_end, INLINE_FN_END);
249 }
```

```
**********************************************************
    5146 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/smatch_var_sym.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Copyright (C) 2013 Oracle.
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, see http://www.gnu.org/copyleft/gpl.txt
 */

#include "smatch.h"

ALLOCATOR(var_sym, "var_sym structs");

struct var_sym *alloc_var_sym(const char *var, struct symbol *sym)
{
        struct var_sym *tmp;

        tmp = __alloc_var_sym(0);
        tmp->var = alloc_string(var);
        tmp->sym = sym;
        return tmp;
}

struct var_sym_list *expr_to_vsl(struct expression *expr)
{
        struct expression *unop;
        struct var_sym_list *ret = NULL;
        char *var;
        struct symbol *sym;

        expr = strip_expr(expr);
        if (!expr)
                return NULL;

        if ((expr->type == EXPR_PREOP && expr->op == '*')) {
                unop = strip_expr(expr->unop);

                if (unop->type == EXPR_SYMBOL)
                        goto one_var;
                return expr_to_vsl(unop);
        }

        if (expr->type == EXPR_BINOP ||
            expr->type == EXPR_LOGICAL ||
            expr->type == EXPR_COMPARE) {
                struct var_sym_list *left, *right;

                left = expr_to_vsl(expr->left);
                right = expr_to_vsl(expr->right);
                ret = combine_var_sym_lists(left, right);
                free_var_syms_and_list(&left);
                free_var_syms_and_list(&right);
```



```
                return ret;
        }

        if (expr->type == EXPR_DEREF)
                return expr_to_vsl(expr->deref);

one_var:
        var = expr_to_var_sym(expr, &sym);
        if (!var || !sym) {
                free_string(var);
                return NULL;
        }
        add_var_sym(&ret, var, sym);
        return ret;
}

int cmp_var_sym(const struct var_sym *a, const struct var_sym *b)
{
        int ret;

        if (a == b)
                return 0;
        if (!b)
                return -1;
        if (!a)
                return 1;

        ret = strcmp(a->var, b->var);
        if (ret < 0)
                return -1;
        if (ret > 0)
                return 1;

        if (!b->sym && a->sym)
                return -1;
        if (!a->sym && b->sym)
                return 1;
        if (a->sym < b->sym)
                return -1;
        if (a->sym > b->sym)
                return 1;

        return 0;
}

void add_var_sym(struct var_sym_list **list, const char *var, struct symbol *sym
{
        struct var_sym *tmp, *new;

        if (in_var_sym_list(*list, var, sym))
                return;
        new = alloc_var_sym(var, sym);

        FOR_EACH_PTR(*list, tmp) {
                if (cmp_var_sym(tmp, new) < 0)
                        continue;
                else if (cmp_var_sym(tmp, new) == 0) {
                        return;
                } else {
                        INSERT_CURRENT(new, tmp);
                        return;
                }
        } END_FOR_EACH_PTR(tmp);
        add_ptr_list(list, new);
}
```

```
127 void add_var_sym_expr(struct var_sym_list **list, struct expression *expr)
128 {
129         char *var;
130         struct symbol *sym;
131
132         var = expr_to_var_sym(expr, &sym);
133         if (!var || !sym)
134                 goto free;
135         add_var_sym(list, var, sym);
136 free:
137         free_string(var);
138 }
139
140 static void free_var_sym(struct var_sym *vs)
141 {
142         free_string(vs->var);
143         __free_var_sym(vs);
144 }
145
146 void del_var_sym(struct var_sym_list **list, const char *var, struct symbol *sym
147 {
148         struct var_sym *tmp;
149
150         FOR_EACH_PTR(*list, tmp) {
151                 if (tmp->sym == sym && strcmp(tmp->var, var) == 0) {
152                         DELETE_CURRENT_PTR(tmp);
153                         free_var_sym(tmp);
154                         return;
155                 }
156         } END_FOR_EACH_PTR(tmp);
157 }
158
159 int in_var_sym_list(struct var_sym_list *list, const char *var, struct symbol *s
160 {
161         struct var_sym *tmp;
162
163         FOR_EACH_PTR(list, tmp) {
164                 if (tmp->sym == sym && strcmp(tmp->var, var) == 0)
165                         return 1;
166         } END_FOR_EACH_PTR(tmp);
167         return 0;
168 }
169
170 struct var_sym_list *clone_var_sym_list(struct var_sym_list *from_vsl)
171 {
172         struct var_sym *tmp, *clone_vs;
173         struct var_sym_list *to_vsl = NULL;
174
175         FOR_EACH_PTR(from_vsl, tmp) {
176                 clone_vs = alloc_var_sym(tmp->var, tmp->sym);
177                 add_ptr_list(&to_vsl, clone_vs);
178         } END_FOR_EACH_PTR(tmp);
179         return to_vsl;
180 }
181
182 void merge_var_sym_list(struct var_sym_list **dest, struct var_sym_list *src)
183 {
184         struct var_sym *tmp;
185
186         FOR_EACH_PTR(src, tmp) {
187                 add_var_sym(dest, tmp->var, tmp->sym);
188         } END_FOR_EACH_PTR(tmp);
189 }
190
191 struct var_sym_list *combine_var_sym_lists(struct var_sym_list *one, struct var_
192 {
```



```
193         struct var_sym_list *to_vsl;
194
195         to_vsl = clone_var_sym_list(one);
196         merge_var_sym_list(&to_vsl, two);
197         return to_vsl;
198 }
199
200 int var_sym_lists_equiv(struct var_sym_list *one, struct var_sym_list *two)
201 {
202         struct var_sym *one_tmp, *two_tmp;
203
204         if (one == two)
205                 return 1;
206
207         if (ptr_list_size((struct ptr_list *)one) != ptr_list_size((struct ptr_l
208                 return 0;
209
210         PREPARE_PTR_LIST(one, one_tmp);
211         PREPARE_PTR_LIST(two, two_tmp);
212         for (;;) {
213                 if (!one_tmp && !two_tmp)
214                         return 1;
215                 if (one_tmp->sym != two_tmp->sym)
216                         return 0;
217                 if (strcmp(one_tmp->var, two_tmp->var) != 0)
218                         return 0;
219                 NEXT_PTR_LIST(one_tmp);
220                 NEXT_PTR_LIST(two_tmp);
221         }
222         FINISH_PTR_LIST(two_tmp);
223         FINISH_PTR_LIST(one_tmp);
224
225         return 1;
226 }
227
228 void free_var_sym_list(struct var_sym_list **list)
229 {
230         __free_ptr_list((struct ptr_list **)list);
231 }
232
233 void free_var_syms_and_list(struct var_sym_list **list)
234 {
235         struct var_sym *tmp;
236
237         FOR_EACH_PTR(*list, tmp) {
238                 free_var_sym(tmp);
239         } END_FOR_EACH_PTR(tmp);
240         free_var_sym_list(list);
241 }
```

```
**********************************************************
    5699 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/sort.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sort_list: a stable sort for lists.
 *
 * Time complexity: O(n*log n)
 *   [assuming limited zero-element fragments]
 *
 * Space complexity: O(1).
 *
 * Stable: yes.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "lib.h"
#include "allocate.h"

#undef PARANOIA
#undef COVERAGE

#ifdef PARANOIA
#include <assert.h>
#else
#define assert(x)
#endif

#ifdef COVERAGE
static unsigned char been_there[256];
#define BEEN_THERE(_c)                                  \
  do {                                                  \
        if (!been_there[_c]) {                          \
                been_there[_c] = 1;                     \
                printf ("Been there: %c\n", _c);        \
        }                                               \
  } while (0)
#else
#define BEEN_THERE(_c) do { } while (0)
#endif

// Sort one fragment.  LIST_NODE_NR (==29) is a bit too high for my
// taste for something this simple.  But, hey, it's O(1).
//
// I would use libc qsort for this, but its comparison function
// gets a pointer indirection extra.
static void array_sort(void **ptr, int nr, int (*cmp)(const void *, const void *
{
        int i;
        for (i = 1; i < nr; i++) {
                void *p = ptr[i];
                if (cmp(ptr[i-1],p) > 0) {
                        int j = i;
                        do {
                                ptr[j] = ptr[j-1];
                                if (!--j)
                                        break;
                        } while (cmp(ptr[j-1], p) > 0);
                        ptr[j] = p;
                }
        }
```



```
}

#ifdef PARANOIA
static void verify_seq_sorted (struct ptr_list *l, int n,
                               int (*cmp)(const void *, const void *))
{
        int i = 0;
        const void *a;
        struct ptr_list *head = l;

        while (l->nr == 0) {
                l = l->next;
                if (--n == 0)
                        return;
                assert (l != head);
        }

        a = l->list[0];
        while (n > 0) {
                const void *b;
                if (++i >= l->nr) {
                        i = 0;
                        l = l->next;
                        n--;
                        assert (l != head || n == 0);
                        continue;
                }
                b = l->list[i];
                assert (cmp (a, b) <= 0);
                a = b;
        }
}
#endif


#define FLUSH_TO(b)                                             \
  do {                                                          \
        int nr = (b)->nr;                                       \
        assert (nbuf >= nr);                                    \
        memcpy ((b)->list, buffer, nr * sizeof (void *));       \
        nbuf -= nr;                                             \
        memmove (buffer, buffer + nr, nbuf * sizeof (void *));  \
  } while (0)

#define DUMP_TO(b)                                              \
  do {                                                          \
        assert (nbuf <= (b)->nr);                               \
        memcpy ((b)->list, buffer, nbuf * sizeof (void *));     \
  } while (0)


// Merge two already-sorted sequences of blocks:
//   (b1_1, ..., b1_n)  and  (b2_1, ..., b2_m)
// Since we may be moving blocks around, we return the new head
// of the merged list.
static struct ptr_list *
merge_block_seqs (struct ptr_list *b1, int n,
                  struct ptr_list *b2, int m,
                  int (*cmp)(const void *, const void *))
{
        int i1 = 0, i2 = 0;
        const void *buffer[2 * LIST_NODE_NR];
        int nbuf = 0;
        struct ptr_list *newhead = b1;

        // printf ("Merging %d blocks at %p with %d blocks at %p\n", n, b1, m, b
```

```
128         // Skip empty blocks in b2.
129         while (b2->nr == 0) {
130                 BEEN_THERE('F');
131                 b2 = b2->next;
132                 if (--m == 0) {
133                         BEEN_THERE('G');
134                         return newhead;
135                 }
136         }
137
138         // Do a quick skip in case entire blocks from b1 are
139         // already less than smallest element in b2.
140         while (b1->nr == 0 ||
141                cmp (PTR_ENTRY(b1, b1->nr - 1), PTR_ENTRY(b2,0)) < 0) {
142                 // printf ("Skipping whole block.\n");
143                 BEEN_THERE('H');
144                 b1 = b1->next;
145                 if (--n == 0) {
146                         BEEN_THERE('I');
147                         return newhead;
148                 }
149         }
150
151         while (1) {
152                 const void *d1 = PTR_ENTRY(b1,i1);
153                 const void *d2 = PTR_ENTRY(b2,i2);
154
155                 assert (i1 >= 0 && i1 < b1->nr);
156                 assert (i2 >= 0 && i2 < b2->nr);
157                 assert (b1 != b2);
158                 assert (n > 0);
159                 assert (m > 0);
160
161                 if (cmp (d1, d2) <= 0) {
162                         BEEN_THERE('J');
163                         buffer[nbuf++] = d1;
164                         // Element from b1 is smaller
165                         if (++i1 >= b1->nr) {
166                                 BEEN_THERE('L');
167                                 FLUSH_TO(b1);
168                                 do {
169                                         b1 = b1->next;
170                                         if (--n == 0) {
171                                                 BEEN_THERE('O');
172                                                 while (b1 != b2) {
173                                                         BEEN_THERE('P');
174                                                         FLUSH_TO(b1);
175                                                         b1 = b1->next;
176                                                 }
177                                                 assert (nbuf == i2);
178                                                 DUMP_TO(b2);
179                                                 return newhead;
180                                         }
181                                 } while (b1->nr == 0);
182                                 i1 = 0;
183                         }
184                 } else {
185                         BEEN_THERE('K');
186                         // Element from b2 is smaller
187                         buffer[nbuf++] = d2;
188                         if (++i2 >= b2->nr) {
189                                 struct ptr_list *l = b2;
190                                 BEEN_THERE('M');
191                                 // OK, we finished with b2.  Pull it out
192                                 // and plug it in before b1.
```



```
194                                 b2 = b2->next;
195                                 b2->prev = l->prev;
196                                 b2->prev->next = b2;
197                                 l->next = b1;
198                                 l->prev = b1->prev;
199                                 l->next->prev = l;
200                                 l->prev->next = l;
201
202                                 if (b1 == newhead) {
203                                         BEEN_THERE('N');
204                                         newhead = l;
205                                 }
206
207                                 FLUSH_TO(l);
208                                 b2 = b2->prev;
209                                 do {
210                                         b2 = b2->next;
211                                         if (--m == 0) {
212                                                 BEEN_THERE('Q');
213                                                 assert (nbuf == i1);
214                                                 DUMP_TO(b1);
215                                                 return newhead;
216                                         }
217                                 } while (b2->nr == 0);
218                                 i2 = 0;
219                         }
220                 }
221         }
222 }
223
224
225 void sort_list(struct ptr_list **plist, int (*cmp)(const void *, const void *))
226 {
227         struct ptr_list *head = *plist, *list = head;
228         int blocks = 1;
229
230         if (!head)
231                 return;
232
233         // Sort all the sub-lists
234         do {
235                 array_sort(list->list, list->nr, cmp);
236 #ifdef PARANOIA
237                 verify_seq_sorted (list, 1, cmp);
238 #endif
239                 list = list->next;
240         } while (list != head);
241
242         // Merge the damn things together
243         while (1) {
244                 struct ptr_list *block1 = head;
245
246                 do {
247                         struct ptr_list *block2 = block1;
248                         struct ptr_list *next, *newhead;
249                         int i;
250
251                         for (i = 0; i < blocks; i++) {
252                                 block2 = block2->next;
253                                 if (block2 == head) {
254                                         if (block1 == head) {
255                                                 BEEN_THERE('A');
256                                                 *plist = head;
257                                                 return;
258                                         }
```

```
259                                         BEEN_THERE('B');
260                                         goto next_pass;
261                                 }
262                         }

264                         next = block2;
265                         for (i = 0; i < blocks; ) {
266                                 next = next->next;
267                                 i++;
268                                 if (next == head) {
269                                         BEEN_THERE('C');
270                                         break;
271                                 }
272                                 BEEN_THERE('D');
273                         }

275                         newhead = merge_block_seqs (block1, blocks,
276                                                     block2, i,
277                                                     cmp);
278 #ifdef PARANOIA
279                         verify_seq_sorted (newhead, blocks + i, cmp);
280 #endif
281                         if (block1 == head) {
282                                 BEEN_THERE('E');
283                                 head = newhead;
284                         }
285                         block1 = next;
286                 } while (block1 != head);
287         next_pass:
288                 blocks <<= 1;
289         }
290 }
```

```
**********************************************************
   27612 Fri Dec 21 15:00:37 2018
new/usr/src/tools/smatch/src/sparse-llvm.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Example usage:
 *      ./sparse-llvm hello.c | llc | as -o hello.o
 */

#include <llvm-c/Core.h>
#include <llvm-c/BitWriter.h>
#include <llvm-c/Analysis.h>
#include <llvm-c/Target.h>

#include <stdbool.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <assert.h>

#include "symbol.h"
#include "expression.h"
#include "linearize.h"
#include "flow.h"

struct function {
        LLVMBuilderRef                  builder;
        LLVMTypeRef                     type;
        LLVMValueRef                    fn;
        LLVMModuleRef                   module;
};

static inline bool symbol_is_fp_type(struct symbol *sym)
{
        if (!sym)
                return false;

        return sym->ctype.base_type == &fp_type;
}

static LLVMTypeRef symbol_type(LLVMModuleRef module, struct symbol *sym);

static LLVMTypeRef func_return_type(LLVMModuleRef module, struct symbol *sym)
{
        return symbol_type(module, sym->ctype.base_type);
}

static LLVMTypeRef sym_func_type(LLVMModuleRef module, struct symbol *sym)
{
        LLVMTypeRef *arg_type;
        LLVMTypeRef func_type;
        LLVMTypeRef ret_type;
        struct symbol *arg;
        int n_arg = 0;

        /* to avoid strangeness with varargs [for now], we build
         * the function and type anew, for each call.  This
         * is probably wrong.  We should look up the
         * symbol declaration info.
         */

        ret_type = func_return_type(module, sym);

        /* count args, build argument type information */
```



```
        FOR_EACH_PTR(sym->arguments, arg) {
                n_arg++;
        } END_FOR_EACH_PTR(arg);

        arg_type = calloc(n_arg, sizeof(LLVMTypeRef));

        int idx = 0;
        FOR_EACH_PTR(sym->arguments, arg) {
                struct symbol *arg_sym = arg->ctype.base_type;

                arg_type[idx++] = symbol_type(module, arg_sym);
        } END_FOR_EACH_PTR(arg);
        func_type = LLVMFunctionType(ret_type, arg_type, n_arg,
                                     sym->variadic);

        return func_type;
}

static LLVMTypeRef sym_array_type(LLVMModuleRef module, struct symbol *sym)
{
        LLVMTypeRef elem_type;
        struct symbol *base_type;

        base_type = sym->ctype.base_type;
        /* empty struct is undefined [6.7.2.1(8)] */
        assert(base_type->bit_size > 0);

        elem_type = symbol_type(module, base_type);
        if (!elem_type)
                return NULL;

        return LLVMArrayType(elem_type, sym->bit_size / base_type->bit_size);
}

#define MAX_STRUCT_MEMBERS 64

static LLVMTypeRef sym_struct_type(LLVMModuleRef module, struct symbol *sym)
{
        LLVMTypeRef elem_types[MAX_STRUCT_MEMBERS];
        struct symbol *member;
        char buffer[256];
        LLVMTypeRef ret;
        unsigned nr = 0;

        snprintf(buffer, sizeof(buffer), "struct.%s", sym->ident ? sym->ident->n
        ret = LLVMStructCreateNamed(LLVMGetGlobalContext(), buffer);
        /* set ->aux to avoid recursion */
        sym->aux = ret;

        FOR_EACH_PTR(sym->symbol_list, member) {
                LLVMTypeRef member_type;

                assert(nr < MAX_STRUCT_MEMBERS);

                member_type = symbol_type(module, member);

                elem_types[nr++] = member_type;
        } END_FOR_EACH_PTR(member);

        LLVMStructSetBody(ret, elem_types, nr, 0 /* packed? */);
        return ret;
}

static LLVMTypeRef sym_union_type(LLVMModuleRef module, struct symbol *sym)
{
        LLVMTypeRef elements;
```

```
127         unsigned union_size;

129         /*
130          * There's no union support in the LLVM API so we treat unions as
131          * opaque structs. The downside is that we lose type information on the
132          * members but as LLVM doesn't care, neither do we.
133          */
134         union_size = sym->bit_size / 8;

136         elements = LLVMArrayType(LLVMInt8Type(), union_size);

138         return LLVMStructType(&elements, 1, 0 /* packed? */);
139 }

141 static LLVMTypeRef sym_ptr_type(LLVMModuleRef module, struct symbol *sym)
142 {
143         LLVMTypeRef type;

145         /* 'void *' is treated like 'char *' */
146         if (is_void_type(sym->ctype.base_type))
147                 type = LLVMInt8Type();
148         else
149                 type = symbol_type(module, sym->ctype.base_type);

151         return LLVMPointerType(type, 0);
152 }

154 static LLVMTypeRef sym_basetype_type(struct symbol *sym)
155 {
156         LLVMTypeRef ret = NULL;

158         if (symbol_is_fp_type(sym)) {
159                 switch (sym->bit_size) {
160                 case 32:
161                         ret = LLVMFloatType();
162                         break;
163                 case 64:
164                         ret = LLVMDoubleType();
165                         break;
166                 case 80:
167                         ret = LLVMX86FP80Type();
168                         break;
169                 default:
170                         die("invalid bit size %d for type %d", sym->bit_size, sy
171                         break;
172                 }
173         } else {
174                 switch (sym->bit_size) {
175                 case -1:
176                         ret = LLVMVoidType();
177                         break;
178                 case 1:
179                         ret = LLVMInt1Type();
180                         break;
181                 case 8:
182                         ret = LLVMInt8Type();
183                         break;
184                 case 16:
185                         ret = LLVMInt16Type();
186                         break;
187                 case 32:
188                         ret = LLVMInt32Type();
189                         break;
190                 case 64:
191                         ret = LLVMInt64Type();
192                         break;
```



```
193                 default:
194                         die("invalid bit size %d for type %d", sym->bit_size, sy
195                         break;
196                 }
197         }

199         return ret;
200 }

202 static LLVMTypeRef symbol_type(LLVMModuleRef module, struct symbol *sym)
203 {
204         LLVMTypeRef ret = NULL;

206         /* don't cache the result for SYM_NODE */
207         if (sym->type == SYM_NODE)
208                 return symbol_type(module, sym->ctype.base_type);

210         if (sym->aux)
211                 return sym->aux;

213         switch (sym->type) {
214         case SYM_BITFIELD:
215         case SYM_ENUM:
216                 ret = symbol_type(module, sym->ctype.base_type);
217                 break;
218         case SYM_BASETYPE:
219                 ret = sym_basetype_type(sym);
220                 break;
221         case SYM_PTR:
222                 ret = sym_ptr_type(module, sym);
223                 break;
224         case SYM_UNION:
225                 ret = sym_union_type(module, sym);
226                 break;
227         case SYM_STRUCT:
228                 ret = sym_struct_type(module, sym);
229                 break;
230         case SYM_ARRAY:
231                 ret = sym_array_type(module, sym);
232                 break;
233         case SYM_FN:
234                 ret = sym_func_type(module, sym);
235                 break;
236         default:
237                 assert(0);
238         }

240         /* cache the result */
241         sym->aux = ret;
242         return ret;
243 }

245 static LLVMTypeRef int_type_by_size(int size)
246 {
247         switch (size) {
248                 case 1:         return LLVMInt1Type();
249                 case 8:         return LLVMInt8Type();
250                 case 16:        return LLVMInt16Type();
251                 case 32:        return LLVMInt32Type();
252                 case 64:        return LLVMInt64Type();

254                 default:
255                         die("invalid bit size %d", size);
256                         break;
257         }
258         return NULL;    /* not reached */
```

```
}

static LLVMTypeRef insn_symbol_type(LLVMModuleRef module, struct instruction *in
{
        if (insn->type)
                return symbol_type(module, insn->type);

        return int_type_by_size(insn->size);
}

static LLVMLinkage data_linkage(struct symbol *sym)
{
        if (sym->ctype.modifiers & MOD_STATIC)
                return LLVMPrivateLinkage;

        return LLVMExternalLinkage;
}

static LLVMLinkage function_linkage(struct symbol *sym)
{
        if (sym->ctype.modifiers & MOD_STATIC)
                return LLVMInternalLinkage;

        return LLVMExternalLinkage;
}

#define MAX_PSEUDO_NAME 64

static void pseudo_name(pseudo_t pseudo, char *buf)
{
        switch (pseudo->type) {
        case PSEUDO_REG:
                snprintf(buf, MAX_PSEUDO_NAME, "R%d", pseudo->nr);
                break;
        case PSEUDO_SYM:
                assert(0);
                break;
        case PSEUDO_VAL:
                assert(0);
                break;
        case PSEUDO_ARG: {
                assert(0);
                break;
        }
        case PSEUDO_PHI:
                snprintf(buf, MAX_PSEUDO_NAME, "PHI%d", pseudo->nr);
                break;
        default:
                assert(0);
        }
}

static LLVMValueRef pseudo_to_value(struct function *fn, struct instruction *ins
{
        LLVMValueRef result = NULL;

        switch (pseudo->type) {
        case PSEUDO_REG:
                result = pseudo->priv;
                break;
        case PSEUDO_SYM: {
                struct symbol *sym = pseudo->sym;
                struct expression *expr;

                assert(sym->bb_target == NULL);
```



```
                expr = sym->initializer;
                if (expr) {
                        switch (expr->type) {
                        case EXPR_STRING: {
                                const char *s = expr->string->data;
                                LLVMValueRef indices[] = { LLVMConstInt(LLVMInt6
                                LLVMValueRef data;

                                data = LLVMAddGlobal(fn->module, LLVMArrayType(L
                                LLVMSetLinkage(data, LLVMPrivateLinkage);
                                LLVMSetGlobalConstant(data, 1);
                                LLVMSetInitializer(data, LLVMConstString(strdup(

                                result = LLVMConstGEP(data, indices, ARRAY_SIZE(
                                break;
                        }
                        case EXPR_SYMBOL: {
                                struct symbol *sym = expr->symbol;

                                result = LLVMGetNamedGlobal(fn->module, show_ide
                                assert(result != NULL);
                                break;
                        }
                        default:
                                assert(0);
                        }
                } else {
                        const char *name = show_ident(sym->ident);
                        LLVMTypeRef type = symbol_type(fn->module, sym);

                        if (LLVMGetTypeKind(type) == LLVMFunctionTypeKind) {
                                result = LLVMGetNamedFunction(fn->module, name);
                                if (!result)
                                        result = LLVMAddFunction(fn->module, nam
                        } else {
                                result = LLVMGetNamedGlobal(fn->module, name);
                                if (!result)
                                        result = LLVMAddGlobal(fn->module, type,
                        }
                }
                break;
        }
        case PSEUDO_VAL:
                result = LLVMConstInt(int_type_by_size(pseudo->size), pseudo->va
                break;
        case PSEUDO_ARG: {
                result = LLVMGetParam(fn->fn, pseudo->nr - 1);
                break;
        }
        case PSEUDO_PHI:
                result = pseudo->priv;
                break;
        case PSEUDO_VOID:
                result = NULL;
                break;
        default:
                assert(0);
        }

        return result;
}

static LLVMValueRef calc_gep(LLVMBuilderRef builder, LLVMValueRef base, LLVMValu
{
        LLVMTypeRef type = LLVMTypeOf(base);
        unsigned int as = LLVMGetPointerAddressSpace(type);
```

```
391         LLVMTypeRef bytep = LLVMPointerType(LLVMInt8Type(), as);
392         LLVMValueRef addr;

394         /* convert base to char* type */
395         base = LLVMBuildPointerCast(builder, base, bytep, "");
396         /* addr = base + off */
397         addr = LLVMBuildInBoundsGEP(builder, base, &off, 1, "");
398         /* convert back to the actual pointer type */
399         addr = LLVMBuildPointerCast(builder, addr, type, "");
400         return addr;
401 }

403 static LLVMRealPredicate translate_fop(int opcode)
404 {
405         static const LLVMRealPredicate trans_tbl[] = {
406                 [OP_SET_EQ]     = LLVMRealOEQ,
407                 [OP_SET_NE]     = LLVMRealUNE,
408                 [OP_SET_LE]     = LLVMRealOLE,
409                 [OP_SET_GE]     = LLVMRealOGE,
410                 [OP_SET_LT]     = LLVMRealOLT,
411                 [OP_SET_GT]     = LLVMRealOGT,
412                 /* Are these used with FP? */
413                 [OP_SET_B]      = LLVMRealOLT,
414                 [OP_SET_A]      = LLVMRealOGT,
415                 [OP_SET_BE]     = LLVMRealOLE,
416                 [OP_SET_AE]     = LLVMRealOGE,
417         };

419         return trans_tbl[opcode];
420 }

422 static LLVMIntPredicate translate_op(int opcode)
423 {
424         static const LLVMIntPredicate trans_tbl[] = {
425                 [OP_SET_EQ]     = LLVMIntEQ,
426                 [OP_SET_NE]     = LLVMIntNE,
427                 [OP_SET_LE]     = LLVMIntSLE,
428                 [OP_SET_GE]     = LLVMIntSGE,
429                 [OP_SET_LT]     = LLVMIntSLT,
430                 [OP_SET_GT]     = LLVMIntSGT,
431                 [OP_SET_B]      = LLVMIntULT,
432                 [OP_SET_A]      = LLVMIntUGT,
433                 [OP_SET_BE]     = LLVMIntULE,
434                 [OP_SET_AE]     = LLVMIntUGE,
435         };

437         return trans_tbl[opcode];
438 }

440 static void output_op_binary(struct function *fn, struct instruction *insn)
441 {
442         LLVMValueRef lhs, rhs, target;
443         char target_name[64];

445         lhs = pseudo_to_value(fn, insn, insn->src1);

447         rhs = pseudo_to_value(fn, insn, insn->src2);

449         pseudo_name(insn->target, target_name);

451         switch (insn->opcode) {
452         /* Binary */
453         case OP_ADD:
454                 if (symbol_is_fp_type(insn->type))
455                         target = LLVMBuildFAdd(fn->builder, lhs, rhs, target_nam
456                 else
```



```
457                         target = LLVMBuildAdd(fn->builder, lhs, rhs, target_name
458                 break;
459         case OP_SUB:
460                 if (symbol_is_fp_type(insn->type))
461                         target = LLVMBuildFSub(fn->builder, lhs, rhs, target_nam
462                 else
463                         target = LLVMBuildSub(fn->builder, lhs, rhs, target_name
464                 break;
465         case OP_MULU:
466                 if (symbol_is_fp_type(insn->type))
467                         target = LLVMBuildFMul(fn->builder, lhs, rhs, target_nam
468                 else
469                         target = LLVMBuildMul(fn->builder, lhs, rhs, target_name
470                 break;
471         case OP_MULS:
472                 assert(!symbol_is_fp_type(insn->type));
473                 target = LLVMBuildMul(fn->builder, lhs, rhs, target_name);
474                 break;
475         case OP_DIVU:
476                 if (symbol_is_fp_type(insn->type))
477                         target = LLVMBuildFDiv(fn->builder, lhs, rhs, target_nam
478                 else
479                         target = LLVMBuildUDiv(fn->builder, lhs, rhs, target_nam
480                 break;
481         case OP_DIVS:
482                 assert(!symbol_is_fp_type(insn->type));
483                 target = LLVMBuildSDiv(fn->builder, lhs, rhs, target_name);
484                 break;
485         case OP_MODU:
486                 assert(!symbol_is_fp_type(insn->type));
487                 target = LLVMBuildURem(fn->builder, lhs, rhs, target_name);
488                 break;
489         case OP_MODS:
490                 assert(!symbol_is_fp_type(insn->type));
491                 target = LLVMBuildSRem(fn->builder, lhs, rhs, target_name);
492                 break;
493         case OP_SHL:
494                 assert(!symbol_is_fp_type(insn->type));
495                 target = LLVMBuildShl(fn->builder, lhs, rhs, target_name);
496                 break;
497         case OP_LSR:
498                 assert(!symbol_is_fp_type(insn->type));
499                 target = LLVMBuildLShr(fn->builder, lhs, rhs, target_name);
500                 break;
501         case OP_ASR:
502                 assert(!symbol_is_fp_type(insn->type));
503                 target = LLVMBuildAShr(fn->builder, lhs, rhs, target_name);
504                 break;
505
506         /* Logical */
507         case OP_AND:
508                 assert(!symbol_is_fp_type(insn->type));
509                 target = LLVMBuildAnd(fn->builder, lhs, rhs, target_name);
510                 break;
511         case OP_OR:
512                 assert(!symbol_is_fp_type(insn->type));
513                 target = LLVMBuildOr(fn->builder, lhs, rhs, target_name);
514                 break;
515         case OP_XOR:
516                 assert(!symbol_is_fp_type(insn->type));
517                 target = LLVMBuildXor(fn->builder, lhs, rhs, target_name);
518                 break;
519         case OP_AND_BOOL: {
520                 LLVMValueRef lhs_nz, rhs_nz;
521                 LLVMTypeRef dst_type;
```

```
523                 lhs_nz = LLVMBuildIsNotNull(fn->builder, lhs, "");
524                 rhs_nz = LLVMBuildIsNotNull(fn->builder, rhs, "");
525                 target = LLVMBuildAnd(fn->builder, lhs_nz, rhs_nz, target_name);

527                 dst_type = insn_symbol_type(fn->module, insn);
528                 target = LLVMBuildZExt(fn->builder, target, dst_type, target_nam
529                 break;
530         }
531         case OP_OR_BOOL: {
532                 LLVMValueRef lhs_nz, rhs_nz;
533                 LLVMTypeRef dst_type;

535                 lhs_nz = LLVMBuildIsNotNull(fn->builder, lhs, "");
536                 rhs_nz = LLVMBuildIsNotNull(fn->builder, rhs, "");
537                 target = LLVMBuildOr(fn->builder, lhs_nz, rhs_nz, target_name);

539                 dst_type = insn_symbol_type(fn->module, insn);
540                 target = LLVMBuildZExt(fn->builder, target, dst_type, target_nam
541                 break;
542         }
543         default:
544                 assert(0);
545                 break;
546         }

548         insn->target->priv = target;
549 }

551 static void output_op_compare(struct function *fn, struct instruction *insn)
552 {
553         LLVMValueRef lhs, rhs, target;
554         char target_name[64];

556         lhs = pseudo_to_value(fn, insn, insn->src1);

558         if (insn->src2->type == PSEUDO_VAL)
559                 rhs = LLVMConstInt(LLVMTypeOf(lhs), insn->src2->value, 1);
560         else
561                 rhs = pseudo_to_value(fn, insn, insn->src2);

563         pseudo_name(insn->target, target_name);

565         LLVMTypeRef dst_type = insn_symbol_type(fn->module, insn);

567         if (LLVMGetTypeKind(LLVMTypeOf(lhs)) == LLVMIntegerTypeKind) {
568                 LLVMIntPredicate op = translate_op(insn->opcode);

570                 target = LLVMBuildICmp(fn->builder, op, lhs, rhs, target_name);
571         } else {
572                 LLVMRealPredicate op = translate_fop(insn->opcode);

574                 target = LLVMBuildFCmp(fn->builder, op, lhs, rhs, target_name);
575         }

577         target = LLVMBuildZExt(fn->builder, target, dst_type, target_name);

579         insn->target->priv = target;
580 }

582 static void output_op_ret(struct function *fn, struct instruction *insn)
583 {
584         pseudo_t pseudo = insn->src;

586         if (pseudo && pseudo != VOID) {
587                 LLVMValueRef result = pseudo_to_value(fn, insn, pseudo);
```



```
589                 LLVMBuildRet(fn->builder, result);
590         } else
591                 LLVMBuildRetVoid(fn->builder);
592 }

594 static LLVMValueRef calc_memop_addr(struct function *fn, struct instruction *ins
595 {
596         LLVMTypeRef int_type, addr_type;
597         LLVMValueRef src, off, addr;
598         unsigned int as;

600         /* int type large enough to hold a pointer */
601         int_type = LLVMIntType(bits_in_pointer);
602         off = LLVMConstInt(int_type, insn->offset, 0);

604         /* convert src to the effective pointer type */
605         src = pseudo_to_value(fn, insn, insn->src);
606         as = LLVMGetPointerAddressSpace(LLVMTypeOf(src));
607         addr_type = LLVMPointerType(insn_symbol_type(fn->module, insn), as);
608         src = LLVMBuildPointerCast(fn->builder, src, addr_type, "");

610         /* addr = src + off */
611         addr = calc_gep(fn->builder, src, off);
612         return addr;
613 }


616 static void output_op_load(struct function *fn, struct instruction *insn)
617 {
618         LLVMValueRef addr, target;

620         addr = calc_memop_addr(fn, insn);

622         /* perform load */
623         target = LLVMBuildLoad(fn->builder, addr, "load_target");

625         insn->target->priv = target;
626 }

628 static void output_op_store(struct function *fn, struct instruction *insn)
629 {
630         LLVMValueRef addr, target, target_in;

632         addr = calc_memop_addr(fn, insn);

634         target_in = pseudo_to_value(fn, insn, insn->target);

636         /* perform store */
637         target = LLVMBuildStore(fn->builder, target_in, addr);

639         insn->target->priv = target;
640 }

642 static LLVMValueRef bool_value(struct function *fn, LLVMValueRef value)
643 {
644         if (LLVMTypeOf(value) != LLVMInt1Type())
645                 value = LLVMBuildIsNotNull(fn->builder, value, "cond");

647         return value;
648 }

650 static void output_op_cbr(struct function *fn, struct instruction *br)
651 {
652         LLVMValueRef cond = bool_value(fn,
653                         pseudo_to_value(fn, br, br->cond));
```

```
655         LLVMBuildCondBr(fn->builder, cond,
656                         br->bb_true->priv,
657                         br->bb_false->priv);
658 }

660 static void output_op_br(struct function *fn, struct instruction *br)
661 {
662         LLVMBuildBr(fn->builder, br->bb_true->priv);
663 }

665 static void output_op_sel(struct function *fn, struct instruction *insn)
666 {
667         LLVMValueRef target, src1, src2, src3;

669         src1 = bool_value(fn, pseudo_to_value(fn, insn, insn->src1));
670         src2 = pseudo_to_value(fn, insn, insn->src2);
671         src3 = pseudo_to_value(fn, insn, insn->src3);

673         target = LLVMBuildSelect(fn->builder, src1, src2, src3, "select");

675         insn->target->priv = target;
676 }

678 static void output_op_switch(struct function *fn, struct instruction *insn)
679 {
680         LLVMValueRef sw_val, target;
681         struct basic_block *def = NULL;
682         struct multijmp *jmp;
683         int n_jmp = 0;

685         FOR_EACH_PTR(insn->multijmp_list, jmp) {
686                 if (jmp->begin == jmp->end) {           /* case N */
687                         n_jmp++;
688                 } else if (jmp->begin < jmp->end) {     /* case M..N */
689                         assert(0);
690                 } else                                  /* default case */
691                         def = jmp->target;
692         } END_FOR_EACH_PTR(jmp);

694         sw_val = pseudo_to_value(fn, insn, insn->target);
695         target = LLVMBuildSwitch(fn->builder, sw_val,
696                                  def ? def->priv : NULL, n_jmp);

698         FOR_EACH_PTR(insn->multijmp_list, jmp) {
699                 if (jmp->begin == jmp->end) {           /* case N */
700                         LLVMAddCase(target,
701                                 LLVMConstInt(LLVMInt32Type(), jmp->begin, 0),
702                                 jmp->target->priv);
703                 } else if (jmp->begin < jmp->end) {     /* case M..N */
704                         assert(0);
705                 }
706         } END_FOR_EACH_PTR(jmp);

708         insn->target->priv = target;
709 }

711 static void output_op_call(struct function *fn, struct instruction *insn)
712 {
713         LLVMValueRef target, func;
714         int n_arg = 0, i;
715         struct pseudo *arg;
716         LLVMValueRef *args;

718         FOR_EACH_PTR(insn->arguments, arg) {
719                 n_arg++;
720         } END_FOR_EACH_PTR(arg);
```



```
722         args = calloc(n_arg, sizeof(LLVMValueRef));

724         i = 0;
725         FOR_EACH_PTR(insn->arguments, arg) {
726                 args[i++] = pseudo_to_value(fn, insn, arg);
727         } END_FOR_EACH_PTR(arg);

729         func = pseudo_to_value(fn, insn, insn->func);
730         target = LLVMBuildCall(fn->builder, func, args, n_arg, "");

732         insn->target->priv = target;
733 }

735 static void output_op_phisrc(struct function *fn, struct instruction *insn)
736 {
737         LLVMValueRef v;
738         struct instruction *phi;

740         assert(insn->target->priv == NULL);

742         /* target = src */
743         v = pseudo_to_value(fn, insn, insn->phi_src);

745         FOR_EACH_PTR(insn->phi_users, phi) {
746                 LLVMValueRef load, ptr;

748                 assert(phi->opcode == OP_PHI);
749                 /* phi must be load from alloca */
750                 load = phi->target->priv;
751                 assert(LLVMGetInstructionOpcode(load) == LLVMLoad);
752                 ptr = LLVMGetOperand(load, 0);
753                 /* store v to alloca */
754                 LLVMBuildStore(fn->builder, v, ptr);
755         } END_FOR_EACH_PTR(phi);
756 }

758 static void output_op_phi(struct function *fn, struct instruction *insn)
759 {
760         LLVMValueRef load = insn->target->priv;

762         /* forward load */
763         assert(LLVMGetInstructionOpcode(load) == LLVMLoad);
764         /* forward load has no parent block */
765         assert(!LLVMGetInstructionParent(load));
766         /* finalize load in current block  */
767         LLVMInsertIntoBuilder(fn->builder, load);
768 }

770 static void output_op_ptrcast(struct function *fn, struct instruction *insn)
771 {
772         LLVMValueRef src, target;
773         char target_name[64];

775         src = insn->src->priv;
776         if (!src)
777                 src = pseudo_to_value(fn, insn, insn->src);

779         pseudo_name(insn->target, target_name);

781         assert(!symbol_is_fp_type(insn->type));

783         target = LLVMBuildBitCast(fn->builder, src, insn_symbol_type(fn->module,

785         insn->target->priv = target;
786 }
```

```
788 static void output_op_cast(struct function *fn, struct instruction *insn, LLVMOp
789 {
790         LLVMValueRef src, target;
791         char target_name[64];

793         src = insn->src->priv;
794         if (!src)
795                 src = pseudo_to_value(fn, insn, insn->src);

797         pseudo_name(insn->target, target_name);

799         assert(!symbol_is_fp_type(insn->type));

801         if (insn->size < LLVMGetIntTypeWidth(LLVMTypeOf(src)))
802                 target = LLVMBuildTrunc(fn->builder, src, insn_symbol_type(fn->m
803         else
804                 target = LLVMBuildCast(fn->builder, op, src, insn_symbol_type(fn

806         insn->target->priv = target;
807 }

809 static void output_insn(struct function *fn, struct instruction *insn)
810 {
811         switch (insn->opcode) {
812         case OP_RET:
813                 output_op_ret(fn, insn);
814                 break;
815         case OP_BR:
816                 output_op_br(fn, insn);
817                 break;
818         case OP_CBR:
819                 output_op_cbr(fn, insn);
820                 break;
821         case OP_SYMADDR:
822                 assert(0);
823                 break;
824         case OP_SETVAL:
825                 assert(0);
826                 break;
827         case OP_SWITCH:
828                 output_op_switch(fn, insn);
829                 break;
830         case OP_COMPUTEDGOTO:
831                 assert(0);
832                 break;
833         case OP_PHISOURCE:
834                 output_op_phisrc(fn, insn);
835                 break;
836         case OP_PHI:
837                 output_op_phi(fn, insn);
838                 break;
839         case OP_LOAD:
840                 output_op_load(fn, insn);
841                 break;
842         case OP_LNOP:
843                 assert(0);
844                 break;
845         case OP_STORE:
846                 output_op_store(fn, insn);
847                 break;
848         case OP_SNOP:
849                 assert(0);
850                 break;
851         case OP_INLINED_CALL:
852                 assert(0);
```



```
853                 break;
854         case OP_CALL:
855                 output_op_call(fn, insn);
856                 break;
857         case OP_CAST:
858                 output_op_cast(fn, insn, LLVMZExt);
859                 break;
860         case OP_SCAST:
861                 output_op_cast(fn, insn, LLVMSExt);
862                 break;
863         case OP_FPCAST:
864                 assert(0);
865                 break;
866         case OP_PTRCAST:
867                 output_op_ptrcast(fn, insn);
868                 break;
869         case OP_BINARY ... OP_BINARY_END:
870                 output_op_binary(fn, insn);
871                 break;
872         case OP_BINCMP ... OP_BINCMP_END:
873                 output_op_compare(fn, insn);
874                 break;
875         case OP_SEL:
876                 output_op_sel(fn, insn);
877                 break;
878         case OP_SLICE:
879                 assert(0);
880                 break;
881         case OP_NOT: {
882                 LLVMValueRef src, target;
883                 char target_name[64];

885                 src = pseudo_to_value(fn, insn, insn->src);

887                 pseudo_name(insn->target, target_name);

889                 target = LLVMBuildNot(fn->builder, src, target_name);

891                 insn->target->priv = target;
892                 break;
893         }
894         case OP_NEG:
895                 assert(0);
896                 break;
897         case OP_CONTEXT:
898                 assert(0);
899                 break;
900         case OP_RANGE:
901                 assert(0);
902                 break;
903         case OP_NOP:
904                 assert(0);
905                 break;
906         case OP_DEATHNOTE:
907                 break;
908         case OP_ASM:
909                 assert(0);
910                 break;
911         case OP_COPY:
912                 assert(0);
913                 break;
914         default:
915                 break;
916         }
917 }
```

```
static void output_bb(struct function *fn, struct basic_block *bb, unsigned long
{
        struct instruction *insn;

        bb->generation = generation;

        FOR_EACH_PTR(bb->insns, insn) {
                if (!insn->bb)
                        continue;

                output_insn(fn, insn);
        }
        END_FOR_EACH_PTR(insn);
}

#define MAX_ARGS        64

static void output_fn(LLVMModuleRef module, struct entrypoint *ep)
{
        unsigned long generation = ++bb_generation;
        struct symbol *sym = ep->name;
        struct symbol *base_type = sym->ctype.base_type;
        struct symbol *ret_type = sym->ctype.base_type->ctype.base_type;
        LLVMTypeRef arg_types[MAX_ARGS];
        LLVMTypeRef return_type;
        struct function function = { .module = module };
        struct basic_block *bb;
        struct symbol *arg;
        const char *name;
        int nr_args = 0;

        FOR_EACH_PTR(base_type->arguments, arg) {
                struct symbol *arg_base_type = arg->ctype.base_type;

                arg_types[nr_args++] = symbol_type(module, arg_base_type);
        } END_FOR_EACH_PTR(arg);

        name = show_ident(sym->ident);

        return_type = symbol_type(module, ret_type);

        function.type = LLVMFunctionType(return_type, arg_types, nr_args, 0);

        function.fn = LLVMAddFunction(module, name, function.type);
        LLVMSetFunctionCallConv(function.fn, LLVMCCallConv);

        LLVMSetLinkage(function.fn, function_linkage(sym));

        function.builder = LLVMCreateBuilder();

        static int nr_bb;

        FOR_EACH_PTR(ep->bbs, bb) {
                if (bb->generation == generation)
                        continue;

                LLVMBasicBlockRef bbr;
                char bbname[32];
                struct instruction *insn;

                sprintf(bbname, "L%d", nr_bb++);
                bbr = LLVMAppendBasicBlock(function.fn, bbname);

                bb->priv = bbr;

                /* allocate alloca for each phi */
```



```
                FOR_EACH_PTR(bb->insns, insn) {
                        LLVMBasicBlockRef entrybbr;
                        LLVMTypeRef phi_type;
                        LLVMValueRef ptr;

                        if (!insn->bb || insn->opcode != OP_PHI)
                                continue;
                        /* insert alloca into entry block */
                        entrybbr = LLVMGetEntryBasicBlock(function.fn);
                        LLVMPositionBuilderAtEnd(function.builder, entrybbr);
                        phi_type = insn_symbol_type(module, insn);
                        ptr = LLVMBuildAlloca(function.builder, phi_type, "");
                        /* emit forward load for phi */
                        LLVMClearInsertionPosition(function.builder);
                        insn->target->priv = LLVMBuildLoad(function.builder, ptr
                } END_FOR_EACH_PTR(insn);
        }
        END_FOR_EACH_PTR(bb);

        FOR_EACH_PTR(ep->bbs, bb) {
                if (bb->generation == generation)
                        continue;

                LLVMPositionBuilderAtEnd(function.builder, bb->priv);

                output_bb(&function, bb, generation);
        }
        END_FOR_EACH_PTR(bb);
}

static LLVMValueRef output_data(LLVMModuleRef module, struct symbol *sym)
{
        struct expression *initializer = sym->initializer;
        LLVMValueRef initial_value;
        LLVMValueRef data;
        const char *name;

        if (initializer) {
                switch (initializer->type) {
                case EXPR_VALUE:
                        initial_value = LLVMConstInt(symbol_type(module, sym), i
                        break;
                case EXPR_SYMBOL: {
                        struct symbol *sym = initializer->symbol;

                        initial_value = LLVMGetNamedGlobal(module, show_ident(sy
                        if (!initial_value)
                                initial_value = output_data(module, sym);
                        break;
                }
                case EXPR_STRING: {
                        const char *s = initializer->string->data;

                        initial_value = LLVMConstString(strdup(s), strlen(s) + 1
                        break;
                }
                default:
                        assert(0);
                }
        } else {
                LLVMTypeRef type = symbol_type(module, sym);

                initial_value = LLVMConstNull(type);
        }

        name = show_ident(sym->ident);
```

```
1052         data = LLVMAddGlobal(module, LLVMTypeOf(initial_value), name);

1054         LLVMSetLinkage(data, data_linkage(sym));
1055         if (sym->ctype.modifiers & MOD_CONST)
1056                 LLVMSetGlobalConstant(data, 1);
1057         if (sym->ctype.modifiers & MOD_TLS)
1058                 LLVMSetThreadLocal(data, 1);
1059         if (sym->ctype.alignment)
1060                 LLVMSetAlignment(data, sym->ctype.alignment);

1062         if (!(sym->ctype.modifiers & MOD_EXTERN))
1063                 LLVMSetInitializer(data, initial_value);

1065         return data;
1066 }

1068 static int is_prototype(struct symbol *sym)
1069 {
1070         if (sym->type == SYM_NODE)
1071                 sym = sym->ctype.base_type;
1072         return sym && sym->type == SYM_FN && !sym->stmt;
1073 }

1075 static int compile(LLVMModuleRef module, struct symbol_list *list)
1076 {
1077         struct symbol *sym;

1079         FOR_EACH_PTR(list, sym) {
1080                 struct entrypoint *ep;
1081                 expand_symbol(sym);

1083                 if (is_prototype(sym))
1084                         continue;

1086                 ep = linearize_symbol(sym);
1087                 if (ep)
1088                         output_fn(module, ep);
1089                 else
1090                         output_data(module, sym);
1091         }
1092         END_FOR_EACH_PTR(sym);

1094         return 0;
1095 }

1097 #ifndef LLVM_DEFAULT_TARGET_TRIPLE
1098 #define LLVM_DEFAULT_TARGET_TRIPLE LLVM_HOSTTRIPLE
1099 #endif

1101 #define X86_LINUX_LAYOUT \
1102         "e-p:32:32:32-i1:8:8-i8:8:8-i16:16:16-i32:32:32-" \
1103         "i64:32:64-f32:32:32-f64:32:64-v64:64:64-v128:128:128-" \
1104         "a0:0:64-f80:32:32-n8:16:32-S128"

1106 #define X86_64_LINUX_LAYOUT \
1107         "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-" \
1108         "i64:64:64-f32:32:32-f64:64:64-v64:64:64-v128:128:128-" \
1109         "a0:0:64-s0:64:64-f80:128:128-n8:16:32:64-S128"

1111 static void set_target(LLVMModuleRef module)
1112 {
1113         char target[] = LLVM_DEFAULT_TARGET_TRIPLE;
1114         const char *arch, *vendor, *os, *env, *layout = NULL;
1115         char triple[256];
```



```
1117         arch = strtok(target, "-");
1118         vendor = strtok(NULL, "-");
1119         os = strtok(NULL, "-");
1120         env = strtok(NULL, "-");

1122         if (!os)
1123                 return;
1124         if (!env)
1125                 env = "unknown";

1127         if (!strcmp(arch, "x86_64") && !strcmp(os, "linux")) {
1128                 if (arch_m64) {
1129                         layout = X86_64_LINUX_LAYOUT;
1130                 } else {
1131                         arch = "i386";
1132                         layout = X86_LINUX_LAYOUT;
1133                 }
1134         }

1136         /* unsupported target */
1137         if (!layout)
1138                 return;

1140         snprintf(triple, sizeof(triple), "%s-%s-%s-%s", arch, vendor, os, env);
1141         LLVMSetTarget(module, triple);
1142         LLVMSetDataLayout(module, layout);
1143 }

1145 int main(int argc, char **argv)
1146 {
1147         struct string_list *filelist = NULL;
1148         struct symbol_list *symlist;
1149         LLVMModuleRef module;
1150         char *file;

1152         symlist = sparse_initialize(argc, argv, &filelist);

1154         module = LLVMModuleCreateWithName("sparse");
1155         set_target(module);

1157         compile(module, symlist);

1159         /* need ->phi_users */
1160         dbg_dead = 1;
1161         FOR_EACH_PTR_NOTAG(filelist, file) {
1162                 symlist = sparse(file);
1163                 if (die_if_error)
1164                         return 1;
1165                 compile(module, symlist);
1166         } END_FOR_EACH_PTR_NOTAG(file);

1168         LLVMVerifyModule(module, LLVMPrintMessageAction, NULL);

1170         LLVMWriteBitcodeToFD(module, STDOUT_FILENO, 0, 0);

1172         LLVMDisposeModule(module);

1174         report_stats();
1175         return 0;
1176 }
```

```
**********************************************************
   15689 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/sparse.1
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
.\" Sparse manpage by Josh Triplett
.TH sparse "1"
.
.SH NAME
sparse \- Semantic Parser for C
.
.SH SYNOPSIS
.B sparse
[\fIWARNING OPTIONS\fR]... \fIfile.c\fR
.
.SH DESCRIPTION
Sparse parses C source and looks for errors, producing warnings on standard
error.
.P
Sparse accepts options controlling the set of warnings to generate.  To turn
on warnings Sparse does not issue by default, use the corresponding warning
option \fB\-Wsomething\fR.  Sparse issues some warnings by default; to turn
off those warnings, pass the negation of the associated warning option,
\fB\-Wno\-something\fR.
.
.SH WARNING OPTIONS
.TP
.B \-Wsparse\-all
Turn on all sparse warnings, except for those explicitly disabled via
\fB\-Wno\-something\fR.
.TP
.B \-Wsparse\-error
Turn all sparse warnings into errors.
.TP
.B \-Waddress\-space
Warn about code which mixes pointers to different address spaces.

Sparse allows an extended attribute
.BI __attribute__((address_space( num )))
on pointers, which designates a pointer target in address space \fInum\fR (a
constant integer).  With \fB\-Waddress\-space\fR, Sparse treats pointers with
identical target types but different address spaces as distinct types.  To
override this warning, such as for functions which convert pointers between
address spaces, use a type that includes \fB__attribute__((force))\fR.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-address\-space\fR.
.
.TP
.B \-Wbitwise
Warn about unsupported operations or type mismatches with restricted integer
types.

Sparse supports an extended attribute, \fB__attribute__((bitwise))\fR, which
creates a new restricted integer type from a base integer type, distinct from
the base integer type and from any other restricted integer type not declared
in the same declaration or \fBtypedef\fR.  For example, this allows programs
to create \fBtypedef\fRs for integer types with specific endianness.  With
\fB-Wbitwise\fR, Sparse will warn on any use of a restricted type in
arithmetic operations other than bitwise operations, and on any conversion of
one restricted type into another, except via a cast that includes
\fB__attribute__((force))\fR.

__bitwise ends up being a "stronger integer separation", one that
doesn't allow you to mix with non-bitwise integers, so now it's much
```



```
harder to lose the type by mistake.

__bitwise is for *unique types* that cannot be mixed with other
types, and that you'd never want to just use as a random integer (the
integer 0 is special, though, and gets silently accepted iirc - it's
kind of like "NULL" for pointers). So "gfp_t" or the "safe endianness"
types would be __bitwise: you can only operate on them by doing
specific operations that know about *that* particular type.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-bitwise\fR.
.
.TP
.B \-Wcast\-to\-as
Warn about casts which add an address space to a pointer type.

A cast that includes \fB__attribute__((force))\fR will suppress this warning.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wcast\-truncate
Warn about casts that truncate constant values.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-cast\-truncate\fR.
.
.TP
.B \-Wconstant\-suffix
Warn if an integer constant is larger than the maximum representable value
of the type indicated by its type suffix (if any). For example, on a
system where ints are 32-bit and longs 64-bit, the constant \fB0x100000000U\fR
is larger than can be represented by an \fBunsigned int\fR but fits in an
\fBunsigned long\fR. So its type is \fBunsigned long\fR but this is not
indicated by its suffix. In this case, the warning could be suppressed by
using the suffix \fBUL\fR: \fB0x100000000UL\fR.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wconstexpr-not-const
Warn if a non-constant expression is encountered when really expecting a
constant expression instead.
Currently, this warns when initializing an object of static storage duration
with an initializer which is not a constant expression.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wcontext
Warn about potential errors in synchronization or other delimited contexts.

Sparse supports several means of designating functions or statements that
delimit contexts, such as synchronization.  Functions with the extended
attribute
.BI __attribute__((context( expression , in_context , out_context ))
require the context \fIexpression\fR (for instance, a lock) to have the value
\fIin_context\fR (a constant nonnegative integer) when called, and return with
the value \fIout_context\fR (a constant nonnegative integer).  For APIs
defined via macros, use the statement form
.BI __context__( expression , in_value , out_value )
in the body of the macro.

With \fB-Wcontext\fR Sparse will warn when it sees a function change the
context without indicating this with a \fBcontext\fR attribute, either by
decreasing a context below zero (such as by releasing a lock without acquiring
```

```
it), or returning with a changed context (such as by acquiring a lock without
releasing it).  Sparse will also warn about blocks of code which may
potentially execute with different contexts.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-context\fR.
.
.TP
.B \-Wdecl
Warn about any non-\fBstatic\fR variable or function definition that has no
previous declaration.

Private symbols (functions and variables) internal to a given source file
should use \fBstatic\fR, to allow additional compiler optimizations, allow
detection of unused symbols, and prevent other code from relying on these
internal symbols.  Public symbols used by other source files will need
declarations visible to those other source files, such as in a header file.
All declarations should fall into one of these two categories.  Thus, with
\fB-Wdecl\fR, Sparse warns about any symbol definition with neither
\fBstatic\fR nor a declaration.  To fix this warning, declare private symbols
\fBstatic\fR, and ensure that the files defining public symbols have the
symbol declarations available first (such as by including the appropriate
header file).

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-decl\fR.
.
.TP
.B \-Wdeclaration-after-statement
Warn about declarations that are not at the start of a block.

These declarations are permitted in C99 but not in C89.

Sparse issues these warnings by default only when the C dialect is
C89 (i.e. -ansi or -std=c89).  To turn them off, use
\fB\-Wno\-declaration\-after\-statement\fR.
.
.TP
.B \-Wdefault\-bitfield\-sign
Warn about any bitfield with no explicit signedness.

Bitfields have no standard-specified default signedness. (C99 6.7.2) A
bitfield without an explicit \fBsigned\fR or \fBunsigned\fR creates a
portability problem for software that relies on the available range of values.
To fix this, specify the bitfield type as \fBsigned\fR or \fBunsigned\fR
explicitly.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wdesignated\-init
Warn about positional initialization of structs marked as requiring designated
initializers.

Sparse allows an attribute
.BI __attribute__((designated_init))
which marks a struct as requiring designated initializers.  Sparse will warn
about positional initialization of a struct variable or struct literal of a
type that has this attribute.

Requiring designated initializers for a particular struct type will insulate
code using that struct type from changes to the layout of the type, avoiding
the need to change initializers for that type unless they initialize a removed
or incompatibly changed field.

Common examples of this type of struct include collections of function pointers
```



```
for the implementations of a class of related operations, for which the default
NULL for an unmentioned field in a designated initializer will correctly
indicate the absence of that operation.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-designated\-init\fR.
.
.TP
.B \-Wdo\-while
Warn about do-while loops that do not delimit the loop body with braces.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wenum\-mismatch
Warn about the use of an expression of an incorrect \fBenum\fR type when
initializing another \fBenum\fR type, assigning to another \fBenum\fR type, or
passing an argument to a function which expects another \fBenum\fR type.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-enum\-mismatch\fR.
.
.TP
.B \-Wempty\-character\-constant
Warn about a constant such as ''.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-empty\-character\-constant\fR.
.
.TP
.B \-Wexternal\-function\-has\-definition
Warn about function definitions that are declared with external linkage.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-external\-function\-has\-definition\fR.
.
.TP
.B \-Winit\-cstring
Warn about initialization of a char array with a too long constant C string.

If the size of the char array and the length of the string are the same,
there is no space for the last nul char of the string in the array:

.nf
char s[3] = "abc";
.fi

If the array is used as a byte array, not as C string, this
warning is just noise. However, if the array is passed to functions
dealing with C string like printf(%s) and strcmp, it may cause a
trouble.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wmemcpy\-max\-count
Warn about call of \fBmemcpy()\fR, \fBmemset()\fR, \fBcopy_from_user()\fR, or
\fBcopy_to_user()\fR with a large compile-time byte count.

Sparse issues these warnings by default. To turn them off, use
\fB\-Wno\-memcpy\-max\-count\fR.

The limit can be changed with \fB\-fmemcpy\-max\-count=COUNT\fR,
the default being \fB100000\fR.
.
.TP
```

```
.B \-Wnon\-ansi\-function\-declaration
Warn about non-ANSI function declarations.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-non\-ansi\-function\-declaration\fR.
.
.TP
.B \-Wnon\-pointer\-null
Warn about the use of 0 as a NULL pointer.

0 has integer type.  NULL has pointer type.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-non\-pointer\-null\fR.
.
.TP
.B \-Wold\-initializer
Warn about the use of the pre-C99 GCC syntax for designated initializers.

C99 provides a standard syntax for designated fields in \fBstruct\fR or
\fBunion\fR initializers:

.nf
struct structname var = { .field = value };
.fi

GCC also has an old, non-standard syntax for designated initializers which
predates C99:

.nf
struct structname var = { field: value };
.fi

Sparse will warn about the use of GCC's non-standard syntax for designated
initializers.  To fix this warning, convert designated initializers to use the
standard C99 syntax.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-old\-initializer\fR.
.
.TP
.B \-Wone\-bit\-signed\-bitfield
Warn about any one-bit \fBsigned\fR bitfields.

A one-bit \fBsigned\fR bitfield can only have the values 0 and -1, or with
some compilers only 0; this results in unexpected behavior for programs which
expected the ability to store 0 and 1.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-one\-bit\-signed\-bitfield\fR.
.
.TP
.B \-Wparen\-string
Warn about the use of a parenthesized string to initialize an array.

Standard C syntax does not permit a parenthesized string as an array
initializer.  GCC allows this syntax as an extension.  With
\fB\-Wparen\-string\fR, Sparse will warn about this syntax.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wpointer\-arith
Warn about anything that depends on the \fBsizeof\fR a void or function type.

C99 does not allow the \fBsizeof\fR operator to be applied to function types
```



```
or to incomplete types such as void. GCC allows \fBsizeof\fR to be applied to
these types as an extension and assigns these types a size of \fI1\fR. With
\fB\-pointer\-arith\fR, Sparse will warn about pointer arithmetic on void
or function pointers, as well as expressions which directly apply the
\fBsizeof\fR operator to void or function types.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wptr\-subtraction\-blows
Warn when subtracting two pointers to a type with a non-power-of-two size.

Subtracting two pointers to a given type gives a difference in terms of the
number of items of that type.  To generate this value, compilers will usually
need to divide the difference by the size of the type, an potentially
expensive operation for sizes other than powers of two.

Code written using pointer subtraction can often use another approach instead,
such as array indexing with an explicit array index variable, which may allow
compilers to generate more efficient code.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wreturn\-void
Warn if a function with return type void returns a void expression.

C99 permits this, and in some cases this allows for more generic code in
macros that use typeof or take a type as a macro argument.  However, some
programs consider this poor style, and those programs can use
\fB\-Wreturn\-void\fR to get warnings about it.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wshadow
Warn when declaring a symbol which shadows a declaration with the same name in
an outer scope.

Such declarations can lead to error-prone code.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wsizeof-bool
Warn when checking the sizeof a _Bool.

C99 does not specify the sizeof a _Bool.  gcc uses 1.

Sparse does not issue these warnings by default.
.
.TP
.B \-Wtransparent\-union
Warn about any declaration using the GCC extension
\fB__attribute__((transparent_union))\fR.

Sparse issues these warnings by default.  To turn them off, use
\fB\-Wno\-transparent\-union\fR.
.
.TP
.B \-Wtypesign
Warn when converting a pointer to an integer type into a pointer to an integer
type with different signedness.

Sparse does not issue these warnings by default.
.
```

```
.TP
.B \-Wundef
Warn about preprocessor conditionals that use the value of an undefined
preprocessor symbol.

Standard C (C99 6.10.1) permits using the value of an undefined preprocessor
symbol in preprocessor conditionals, and specifies it has a value of 0.
However, this behavior can lead to subtle errors.

Sparse does not issue these warnings by default.
.
.SH MISC OPTIONS
.TP
.B \-gcc-base-dir \fIdir\fR
Look for compiler-provided system headers in \fIdir\fR/include/ and \fIdir\fR/in
.
.TP
.B \-multiarch-dir \fIdir\fR
Look for system headers in the multiarch subdirectory \fIdir\fR.
The \fIdir\fR name would normally take the form of the target's
normalized GNU triplet. (e.g. i386-linux-gnu).
.
.SH DEBUG OPTIONS
.TP
.B \-fdump-linearize[=only]
Dump the IR code of a function directly after its linearization,
before any simplifications are made. If the argument \fB=only\fR is
also given no further processing is done on the function.
.
.B \-fmem-report
Report some statistics about memory allocation used by the tool.
.
.SH OTHER OPTIONS
.TP
.B \-fmemcpy-max-count=COUNT
Set the limit for the warnings given by \fB-Wmemcpy-max-count\fR.
A COUNT of 0, useless in itself, will effectively disable the warning.
The default limit is 100000.
.
.TP
.B \-ftabstop=WIDTH
Set the distance between tab stops.  This helps sparse report correct
column numbers in warnings or errors.  If the value is less than 1 or
greater than 100, the option is ignored.  The default is 8.
.
.SH SEE ALSO
.BR cgcc (1)
.
.SH HOMEPAGE
http://www.kernel.org/pub/software/devel/sparse/
.
.SH MAILING LIST
linux-sparse@vger.kernel.org
.
.SH MAINTAINER
Christopher Li <sparse@chrisli.org>
```

```
**********************************************************
    7672 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/sparse.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Example trivial client program that uses the sparse library
 * to tokenize, preprocess and parse a C file, and prints out
 * the results.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"
#include "linearize.h"

static int context_increase(struct basic_block *bb, int entry)
{
        int sum = 0;
        struct instruction *insn;

        FOR_EACH_PTR(bb->insns, insn) {
                int val;
                if (insn->opcode != OP_CONTEXT)
                        continue;
                val = insn->increment;
                if (insn->check) {
                        int current = sum + entry;
                        if (!val) {
                                if (!current)
                                        continue;
                        } else if (current >= val)
                                continue;
                        warning(insn->pos, "context check failure");
```



```
                        continue;
                sum += val;
        } END_FOR_EACH_PTR(insn);
        return sum;
}

static int imbalance(struct entrypoint *ep, struct basic_block *bb, int entry, i
{
        if (Wcontext) {
                struct symbol *sym = ep->name;
                warning(bb->pos, "context imbalance in '%s' - %s", show_ident(sy
        }
        return -1;
}

static int check_bb_context(struct entrypoint *ep, struct basic_block *bb, int e

static int check_children(struct entrypoint *ep, struct basic_block *bb, int ent
{
        struct instruction *insn;
        struct basic_block *child;

        insn = last_instruction(bb->insns);
        if (!insn)
                return 0;
        if (insn->opcode == OP_RET)
                return entry != exit ? imbalance(ep, bb, entry, exit, "wrong cou

        FOR_EACH_PTR(bb->children, child) {
                if (check_bb_context(ep, child, entry, exit))
                        return -1;
        } END_FOR_EACH_PTR(child);
        return 0;
}

static int check_bb_context(struct entrypoint *ep, struct basic_block *bb, int e
{
        if (!bb)
                return 0;
        if (bb->context == entry)
                return 0;

        /* Now that's not good.. */
        if (bb->context >= 0)
                return imbalance(ep, bb, entry, bb->context, "different lock con

        bb->context = entry;
        entry += context_increase(bb, entry);
        if (entry < 0)
                return imbalance(ep, bb, entry, exit, "unexpected unlock");

        return check_children(ep, bb, entry, exit);
}

static void check_cast_instruction(struct instruction *insn)
{
        struct symbol *orig_type = insn->orig_type;
        if (orig_type) {
                int old = orig_type->bit_size;
                int new = insn->size;
                int oldsigned = (orig_type->ctype.modifiers & MOD_SIGNED) != 0;
                int newsigned = insn->opcode == OP_SCAST;

                if (new > old) {
                        if (oldsigned == newsigned)
```

```
127                                 return;
128                         if (newsigned)
129                                 return;
130                         warning(insn->pos, "cast loses sign");
131                         return;
132                 }
133                 if (new < old) {
134                         warning(insn->pos, "cast drops bits");
135                         return;
136                 }
137                 if (oldsigned == newsigned) {
138                         warning(insn->pos, "cast wasn't removed");
139                         return;
140                 }
141                 warning(insn->pos, "cast changes sign");
142         }
143 }

145 static void check_range_instruction(struct instruction *insn)
146 {
147         warning(insn->pos, "value out of range");
148 }

150 static void check_byte_count(struct instruction *insn, pseudo_t count)
151 {
152         if (!count)
153                 return;
154         if (count->type == PSEUDO_VAL) {
155                 unsigned long long val = count->value;
156                 if (Wmemcpy_max_count && val > fmemcpy_max_count)
157                         warning(insn->pos, "%s with byte count of %llu",
158                                 show_ident(insn->func->sym->ident), val);
159                 return;
160         }
161         /* OK, we could try to do the range analysis here */
162 }

164 static pseudo_t argument(struct instruction *call, unsigned int argno)
165 {
166         pseudo_t args[8];
167         struct ptr_list *arg_list = (struct ptr_list *) call->arguments;

169         argno--;
170         if (linearize_ptr_list(arg_list, (void *)args, 8) > argno)
171                 return args[argno];
172         return NULL;
173 }

175 static void check_memset(struct instruction *insn)
176 {
177         check_byte_count(insn, argument(insn, 3));
178 }

180 #define check_memcpy check_memset
181 #define check_ctu check_memset
182 #define check_cfu check_memset

184 struct checkfn {
185         struct ident *id;
186         void (*check)(struct instruction *insn);
187 };

189 static void check_call_instruction(struct instruction *insn)
190 {
191         pseudo_t fn = insn->func;
192         struct ident *ident;
```



```
193         static const struct checkfn check_fn[] = {
194                 { &memset_ident, check_memset },
195                 { &memcpy_ident, check_memcpy },
196                 { &copy_to_user_ident, check_ctu },
197                 { &copy_from_user_ident, check_cfu },
198         };
199         int i;

201         if (fn->type != PSEUDO_SYM)
202                 return;
203         ident = fn->sym->ident;
204         if (!ident)
205                 return;
206         for (i = 0; i < ARRAY_SIZE(check_fn); i++) {
207                 if (check_fn[i].id != ident)
208                         continue;
209                 check_fn[i].check(insn);
210                 break;
211         }
212 }

214 static void check_one_instruction(struct instruction *insn)
215 {
216         switch (insn->opcode) {
217         case OP_CAST: case OP_SCAST:
218                 if (verbose)
219                         check_cast_instruction(insn);
220                 break;
221         case OP_RANGE:
222                 check_range_instruction(insn);
223                 break;
224         case OP_CALL:
225                 check_call_instruction(insn);
226                 break;
227         default:
228                 break;
229         }
230 }

232 static void check_bb_instructions(struct basic_block *bb)
233 {
234         struct instruction *insn;
235         FOR_EACH_PTR(bb->insns, insn) {
236                 if (!insn->bb)
237                         continue;
238                 check_one_instruction(insn);
239         } END_FOR_EACH_PTR(insn);
240 }

242 static void check_instructions(struct entrypoint *ep)
243 {
244         struct basic_block *bb;
245         FOR_EACH_PTR(ep->bbs, bb) {
246                 check_bb_instructions(bb);
247         } END_FOR_EACH_PTR(bb);
248 }

250 static void check_context(struct entrypoint *ep)
251 {
252         struct symbol *sym = ep->name;
253         struct context *context;
254         unsigned int in_context = 0, out_context = 0;

256         if (Wuninitialized && verbose && ep->entry->bb->needs) {
257                 pseudo_t pseudo;
258                 FOR_EACH_PTR(ep->entry->bb->needs, pseudo) {
```

```
259                         if (pseudo->type != PSEUDO_ARG)
260                                 warning(sym->pos, "%s: possible uninitialized va
261                                         show_ident(sym->ident), show_pseudo(pseu
262                 } END_FOR_EACH_PTR(pseudo);
263         }

265         check_instructions(ep);

267         FOR_EACH_PTR(sym->ctype.contexts, context) {
268                 in_context += context->in;
269                 out_context += context->out;
270         } END_FOR_EACH_PTR(context);
271         check_bb_context(ep, ep->entry->bb, in_context, out_context);
272 }

274 static void check_symbols(struct symbol_list *list)
275 {
276         struct symbol *sym;

278         FOR_EACH_PTR(list, sym) {
279                 struct entrypoint *ep;

281                 expand_symbol(sym);
282                 ep = linearize_symbol(sym);
283                 if (ep) {
284                         if (dbg_entry)
285                                 show_entry(ep);

287                         check_context(ep);
288                 }
289         } END_FOR_EACH_PTR(sym);

291         if (Wsparse_error && die_if_error)
292                 exit(1);
293 }

295 int main(int argc, char **argv)
296 {
297         struct string_list *filelist = NULL;
298         char *file;

300         // Expand, linearize and show it.
301         check_symbols(sparse_initialize(argc, argv, &filelist));
302         FOR_EACH_PTR_NOTAG(filelist, file) {
303                 check_symbols(sparse(file));
304         } END_FOR_EACH_PTR_NOTAG(file);

306         report_stats();
307         return 0;
308 }
```

```
**********************************************************
     175 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/sparse.pc.in
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
prefix=@prefix@
libdir=@libdir@
includedir=@includedir@

Name: Sparse
Description: Semantic parser for C
Version: @version@
Libs: -L${libdir} -lsparse
Cflags: -I${includedir}
```

```
**********************************************************
     776 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/sparsec
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/sh
#
# GCC compatible C compiler based on Sparse LLVM

set +e

SPARSEOPTS=""
DIRNAME=`dirname $0`

NEED_LINK=1

if [ $# -eq 0 ]; then
  echo "`basename $0`: no input files"
  exit 1
fi

while [ $# -gt 0 ]; do
        case $1 in
        '-o')
                OUTFILE=$2
                shift
                ;;
        '-c')
                NEED_LINK=0
                ;;
        *)
                SPARSEOPTS="$SPARSEOPTS $1 " ;;
        esac
        shift
done

TMPLLVM=`mktemp -t tmp.XXXXXX`".llvm"
TMPFILE=`mktemp -t tmp.XXXXXX`".o"

$DIRNAME/sparse-llvm $SPARSEOPTS > $TMPLLVM

LLC='"${LLVM_CONFIG:-llvm-config}" --bindir`/llc

$LLC -o - $TMPLLVM | as -o $TMPFILE

if [ $NEED_LINK -eq 1 ]; then
        if [ -z $OUTFILE ]; then
                OUTFILE=a.out
        fi
        gcc $TMPFILE -o $OUTFILE
else
        if [ -z $OUTFILE ]; then
                echo "`basename $0`: no output file"
                exit 1
        fi
        mv $TMPFILE $OUTFILE
fi

rm -f $TMPLLVM
```

```
**********************************************************
     195 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/sparsei
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/sh

set +e

DIRNAME=`dirname $0`
LLI=`"${LLVM_CONFIG:-llvm-config}" --bindir`/lli

if [ $# -eq 0 ]; then
  echo "`basename $0`: no input files"
  exit 1
fi

$DIRNAME/sparse-llvm $@ | $LLI
```

```
**********************************************************
    1714 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/stats.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "allocate.h"
#include "linearize.h"
#include "storage.h"

__DECLARE_ALLOCATOR(struct ptr_list, ptrlist);


typedef void (*get_t)(struct allocator_stats*);

static void show_stats(get_t get, struct allocator_stats * tot)
{
        struct allocator_stats x;

        if (get)
                get(&x);
        else
                x = *tot;
        fprintf(stderr, "%16s: %8d, %10ld, %10ld, %6.2f%%, %8.2f\n",
                x.name, x.allocations, x.useful_bytes, x.total_bytes,
                100 * (double) x.useful_bytes / (x.total_bytes ? : 1),
                (double) x.useful_bytes / (x.allocations ? : 1));

        tot->allocations += x.allocations;
        tot->useful_bytes += x.useful_bytes;
        tot->total_bytes += x.total_bytes;
}

void show_allocation_stats(void)
{
        struct allocator_stats tot = { .name = "total", };

        fprintf(stderr, "%16s: %8s, %10s, %10s, %7s, %8s\n", "allocator", "alloc
                "bytes", "total", "%usage", "average");
        show_stats(get_token_stats, &tot);
        show_stats(get_ident_stats, &tot);
        show_stats(get_symbol_stats, &tot);
        show_stats(get_expression_stats, &tot);
        show_stats(get_statement_stats, &tot);
        show_stats(get_scope_stats, &tot);
        show_stats(get_basic_block_stats, &tot);
        show_stats(get_instruction_stats, &tot);
        show_stats(get_pseudo_stats, &tot);
        show_stats(get_pseudo_user_stats, &tot);
        show_stats(get_ptrlist_stats, &tot);
        show_stats(get_multijmp_stats, &tot);
        show_stats(get_asm_rules_stats, &tot);
        show_stats(get_asm_constraint_stats, &tot);
        show_stats(get_context_stats, &tot);
        show_stats(get_string_stats, &tot);
        show_stats(get_bytes_stats, &tot);
        //show_stats(get_storage_stats, &tot);
        //show_stats(get_storage_hash_stats, &tot);

        show_stats(NULL, &tot);
}

void report_stats(void)
{
        if (fmem_report)
```



```
                show_allocation_stats();
}
```

```
**********************************************************
    7592 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/storage.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Storage - associate pseudos with "storage" that keeps them alive
 * between basic blocks.  The aim is to be able to turn as much of
 * the global storage allocation problem as possible into a local
 * per-basic-block one.
 *
 * Copyright (C) 2004 Linus Torvalds
 */
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "symbol.h"
#include "expression.h"
#include "linearize.h"
#include "storage.h"

ALLOCATOR(storage, "storages");
ALLOCATOR(storage_hash, "storage hash");

#define MAX_STORAGE_HASH 64
static struct storage_hash_list *storage_hash_table[MAX_STORAGE_HASH];

static inline unsigned int storage_hash(struct basic_block *bb, pseudo_t pseudo,
{
        unsigned hash = hashval(bb) + hashval(pseudo) + hashval(inout);
        hash += hash / MAX_STORAGE_HASH;
        return hash & (MAX_STORAGE_HASH-1);
}

static int hash_list_cmp(const void *_a, const void *_b)
{
        const struct storage_hash *a = _a;
        const struct storage_hash *b = _b;
        if (a->pseudo != b->pseudo)
                return a->pseudo < b->pseudo ? -1 : 1;
        return 0;
}

static void sort_hash_list(struct storage_hash_list **listp)
{
        sort_list((struct ptr_list **)listp, hash_list_cmp);
}

struct storage_hash_list *gather_storage(struct basic_block *bb, enum inout_enum
{
        int i;
        struct storage_hash *entry, *prev;
        struct storage_hash_list *list = NULL;

        for (i = 0; i < MAX_STORAGE_HASH; i++) {
                struct storage_hash *hash;
                FOR_EACH_PTR(storage_hash_table[i], hash) {
                        if (hash->bb == bb && hash->inout == inout)
                                add_ptr_list(&list, hash);
                } END_FOR_EACH_PTR(hash);
        }
        sort_hash_list(&list);

        prev = NULL;
```



```
        FOR_EACH_PTR(list, entry) {
                if (prev && entry->pseudo == prev->pseudo) {
                        assert(entry == prev);
                        DELETE_CURRENT_PTR(entry);
                }
                prev = entry;
        } END_FOR_EACH_PTR(entry);
        PACK_PTR_LIST(&list);
        return list;
}

static void name_storage(void)
{
        int i;
        int name = 0;

        for (i = 0; i < MAX_STORAGE_HASH; i++) {
                struct storage_hash *hash;
                FOR_EACH_PTR(storage_hash_table[i], hash) {
                        struct storage *storage = hash->storage;
                        if (storage->name)
                                continue;
                        storage->name = ++name;
                } END_FOR_EACH_PTR(hash);
        }
}

struct storage *lookup_storage(struct basic_block *bb, pseudo_t pseudo, enum ino
{
        struct storage_hash_list *list = storage_hash_table[storage_hash(bb,pseu
        struct storage_hash *hash;

        FOR_EACH_PTR(list, hash) {
                if (hash->bb == bb && hash->pseudo == pseudo && hash->inout == i
                        return hash->storage;
        } END_FOR_EACH_PTR(hash);
        return NULL;
}

void add_storage(struct storage *storage, struct basic_block *bb, pseudo_t pseud
{
        struct storage_hash_list **listp = storage_hash_table + storage_hash(bb,
        struct storage_hash *hash = alloc_storage_hash(storage);

        hash->bb = bb;
        hash->pseudo = pseudo;
        hash->inout = inout;

        add_ptr_list(listp, hash);
}


static int storage_hash_cmp(const void *_a, const void *_b)
{
        const struct storage_hash *a = _a;
        const struct storage_hash *b = _b;
        struct storage *aa = a->storage;
        struct storage *bb = b->storage;

        if (a->bb != b->bb)
                return a->bb < b->bb ? -1 : 1;
        if (a->inout != b->inout)
                return a->inout < b->inout ? -1 : 1;
        if (aa->type != bb->type)
                return aa->type < bb->type ? -1 : 1;
        if (aa->regno != bb->regno)
```

```
127                 return aa->regno < bb->regno ? -1 : 1;
128         return 0;
129 }

131 static void vrfy_storage(struct storage_hash_list **listp)
132 {
133         struct storage_hash *entry, *last;

135         sort_list((struct ptr_list **)listp, storage_hash_cmp);
136         last = NULL;
137         FOR_EACH_PTR(*listp, entry) {
138                 if (last) {
139                         struct storage *a = last->storage;
140                         struct storage *b = entry->storage;
141                         if (a == b)
142                                 continue;
143                         if (last->bb == entry->bb
144                             && last->inout == entry->inout
145                             && a->type != REG_UDEF
146                             && a->type == b->type
147                             && a->regno == b->regno) {
148                                 printf("\t BAD: same storage as %s in %p: %s (%s
149                                         last->inout == STOR_IN ? "input" : "outp
150                                         last->bb,
151                                         show_storage(a),
152                                         show_pseudo(last->pseudo),
153                                         show_pseudo(entry->pseudo));
154                         }
155                 }
156                 last = entry;
157         } END_FOR_EACH_PTR(entry);
158 }

160 void free_storage(void)
161 {
162         int i;

164         for (i = 0; i < MAX_STORAGE_HASH; i++) {
165                 vrfy_storage(storage_hash_table + i);
166                 free_ptr_list(storage_hash_table + i);
167         }
168 }

170 const char *show_storage(struct storage *s)
171 {
172         static char buffer[1024];
173         if (!s)
174                 return "none";
175         switch (s->type) {
176         case REG_REG:
177                 sprintf(buffer, "reg%d (%d)", s->regno, s->name);
178                 break;
179         case REG_STACK:
180                 sprintf(buffer, "%d(SP) (%d)", s->offset, s->name);
181                 break;
182         case REG_ARG:
183                 sprintf(buffer, "ARG%d (%d)", s->regno, s->name);
184                 break;
185         default:
186                 sprintf(buffer, "%d:%d (%d)", s->type, s->regno, s->name);
187                 break;
188         }
189         return buffer;
190 }

192 /*
```



```
193  * Combine two storage allocations into one.
194  *
195  * We just randomly pick one over the other, and replace
196  * the other uses.
197  */
198 static struct storage * combine_storage(struct storage *src, struct storage *dst
199 {
200         struct storage **usep;

202         /* Remove uses of "src_storage", replace with "dst" */
203         FOR_EACH_PTR(src->users, usep) {
204                 assert(*usep == src);
205                 *usep = dst;
206                 add_ptr_list(&dst->users, usep);
207         } END_FOR_EACH_PTR(usep);

209         /* Mark it unused */
210         src->type = REG_BAD;
211         src->users = NULL;
212         return dst;
213 }

215 static void set_up_bb_storage(struct basic_block *bb)
216 {
217         struct basic_block *child;

219         FOR_EACH_PTR(bb->children, child) {
220                 pseudo_t pseudo;
221                 FOR_EACH_PTR(child->needs, pseudo) {
222                         struct storage *child_in, *parent_out;

224                         parent_out = lookup_storage(bb, pseudo, STOR_OUT);
225                         child_in = lookup_storage(child, pseudo, STOR_IN);

227                         if (parent_out) {
228                                 if (!child_in) {
229                                         add_storage(parent_out, child, pseudo, S
230                                         continue;
231                                 }
232                                 if (parent_out == child_in)
233                                         continue;
234                                 combine_storage(parent_out, child_in);
235                                 continue;
236                         }
237                         if (child_in) {
238                                 add_storage(child_in, bb, pseudo, STOR_OUT);
239                                 continue;
240                         }
241                         parent_out = alloc_storage();
242                         add_storage(parent_out, bb, pseudo, STOR_OUT);
243                         add_storage(parent_out, child, pseudo, STOR_IN);
244                 } END_FOR_EACH_PTR(pseudo);
245         } END_FOR_EACH_PTR(child);
246 }

248 static void set_up_argument_storage(struct entrypoint *ep, struct basic_block *b
249 {
250         pseudo_t arg;

252         FOR_EACH_PTR(bb->needs, arg) {
253                 struct storage *storage = alloc_storage();

255                 /* FIXME! Totally made-up argument passing conventions */
256                 if (arg->type == PSEUDO_ARG) {
257                         storage->type = REG_ARG;
258                         storage->regno = arg->nr;
```

```
259                 }
260                 add_storage(storage, bb, arg, STOR_IN);
261         } END_FOR_EACH_PTR(arg);
262 }

264 /*
265  * One phi-source may feed multiple phi nodes. If so, combine
266  * the storage output for this bb into one entry to reduce
267  * storage pressure.
268  */
269 static void combine_phi_storage(struct basic_block *bb)
270 {
271         struct instruction *insn;
272         FOR_EACH_PTR(bb->insns, insn) {
273                 struct instruction *phi;
274                 struct storage *last;

276                 if (!insn->bb || insn->opcode != OP_PHISOURCE)
277                         continue;
278                 last = NULL;
279                 FOR_EACH_PTR(insn->phi_users, phi) {
280                         struct storage *storage = lookup_storage(bb, phi->target
281                         if (!storage) {
282                                 DELETE_CURRENT_PTR(phi);
283                                 continue;
284                         }
285                         if (last && storage != last)
286                                 storage = combine_storage(storage, last);
287                         last = storage;
288                 } END_FOR_EACH_PTR(phi);
289                 PACK_PTR_LIST(&insn->phi_users);
290         } END_FOR_EACH_PTR(insn);
291 }

293 void set_up_storage(struct entrypoint *ep)
294 {
295         struct basic_block *bb;

297         /* First set up storage for the incoming arguments */
298         set_up_argument_storage(ep, ep->entry->bb);

300         /* Then do a list of all the inter-bb storage */
301         FOR_EACH_PTR(ep->bbs, bb) {
302                 set_up_bb_storage(bb);
303                 combine_phi_storage(bb);
304         } END_FOR_EACH_PTR(bb);

306         name_storage();
307 }
```

```
**********************************************************
    1749 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/storage.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef STORAGE_H
#define STORAGE_H

#include "allocate.h"
#include "lib.h"

/*
 * The "storage" that underlies an incoming/outgoing pseudo. It's
 * basically the backing store for a pseudo, and may be a real hardware
 * register, a stack slot or a static symbol. Or nothing at all,
 * since some pseudos can just be recalculated on the fly.
 */
enum storage_type {
        REG_UDEF,
        REG_REG,
        REG_STACK,
        REG_FRAME,
        REG_SYM,
        REG_ARG,
        REG_BAD,
};

enum inout_enum {
        STOR_IN,
        STOR_OUT
};

struct storage;
DECLARE_PTR_LIST(storage_ptr_list, struct storage *);

struct storage {
        enum storage_type type;
        int name;
        struct storage_ptr_list *users;
        union {
                int regno;
                int offset;
                struct symbol *sym;
        };
};

DECLARE_PTR_LIST(storage_list, struct storage);

struct storage_hash {
        struct basic_block *bb;
        pseudo_t pseudo;
        enum inout_enum inout;
        struct storage *storage;
        unsigned long flags;
};

DECLARE_PTR_LIST(storage_hash_list, struct storage_hash);

extern struct storage_hash_list *gather_storage(struct basic_block *, enum inout
extern void free_storage(void);
extern const char *show_storage(struct storage *);
extern void set_up_storage(struct entrypoint *);
struct storage *lookup_storage(struct basic_block *, pseudo_t, enum inout_enum);
void add_storage(struct storage *, struct basic_block *, pseudo_t, enum inout_en
```



```
DECLARE_ALLOCATOR(storage);
DECLARE_ALLOCATOR(storage_hash);

static inline struct storage *alloc_storage(void)
{
        return __alloc_storage(0);
}

static inline struct storage_hash *alloc_storage_hash(struct storage *s)
{
        struct storage_hash *entry = __alloc_storage_hash(0);
        struct storage **usep = &entry->storage;

        *usep = s;
        add_ptr_list(&s->users, usep);
        return entry;
}

#endif /* STORAGE_H */
```

```
**********************************************************
   22339 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/symbol.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Symbol lookup and handling.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "scope.h"
#include "expression.h"

#include "target.h"

/*
 * Secondary symbol list for stuff that needs to be output because it
 * was used.
 */
struct symbol_list *translation_unit_used_list = NULL;

/*
 * If the symbol is an inline symbol, add it to the list of symbols to parse
 */
void access_symbol(struct symbol *sym)
{
        if (sym->ctype.modifiers & MOD_INLINE) {
                if (!(sym->ctype.modifiers & MOD_ACCESSED)) {
                        add_symbol(&translation_unit_used_list, sym);
                        sym->ctype.modifiers |= MOD_ACCESSED;
                }
        }
}

struct symbol *lookup_symbol(struct ident *ident, enum namespace ns)
{
        struct symbol *sym;
```



```
        for (sym = ident->symbols; sym; sym = sym->next_id) {
                if (sym->namespace & ns) {
                        sym->used = 1;
                        return sym;
                }
        }
        return NULL;
}

struct context *alloc_context(void)
{
        return __alloc_context(0);
}

struct symbol *alloc_symbol(struct position pos, int type)
{
        struct symbol *sym = __alloc_symbol(0);
        sym->type = type;
        sym->pos = pos;
        sym->endpos.type = 0;
        return sym;
}

struct struct_union_info {
        unsigned long max_align;
        unsigned long bit_size;
        int align_size;
};

/*
 * Unions are fairly easy to lay out ;)
 */
static void lay_out_union(struct symbol *sym, struct struct_union_info *info)
{
        examine_symbol_type(sym);

        // Unnamed bitfields do not affect alignment.
        if (sym->ident || !is_bitfield_type(sym)) {
                if (sym->ctype.alignment > info->max_align)
                        info->max_align = sym->ctype.alignment;
        }

        if (sym->bit_size > info->bit_size)
                info->bit_size = sym->bit_size;

        sym->offset = 0;
}

static int bitfield_base_size(struct symbol *sym)
{
        if (sym->type == SYM_NODE)
                sym = sym->ctype.base_type;
        if (sym->type == SYM_BITFIELD)
                sym = sym->ctype.base_type;
        return sym->bit_size;
}

/*
 * Structures are a bit more interesting to lay out
 */
static void lay_out_struct(struct symbol *sym, struct struct_union_info *info)
{
        unsigned long bit_size, align_bit_mask;
        int base_size;
```

```
127         examine_symbol_type(sym);

129         // Unnamed bitfields do not affect alignment.
130         if (sym->ident || !is_bitfield_type(sym)) {
131                 if (sym->ctype.alignment > info->max_align)
132                         info->max_align = sym->ctype.alignment;
133         }

135         bit_size = info->bit_size;
136         base_size = sym->bit_size;

138         /*
139          * Unsized arrays cause us to not align the resulting
140          * structure size
141          */
142         if (base_size < 0) {
143                 info->align_size = 0;
144                 base_size = 0;
145         }

147         align_bit_mask = bytes_to_bits(sym->ctype.alignment) - 1;

149         /*
150          * Bitfields have some very special rules..
151          */
152         if (is_bitfield_type (sym)) {
153                 unsigned long bit_offset = bit_size & align_bit_mask;
154                 int room = bitfield_base_size(sym) - bit_offset;
155                 // Zero-width fields just fill up the unit.
156                 int width = base_size ? : (bit_offset ? room : 0);

158                 if (width > room) {
159                         bit_size = (bit_size + align_bit_mask) & ~align_bit_mask
160                         bit_offset = 0;
161                 }
162                 sym->offset = bits_to_bytes(bit_size - bit_offset);
163                 sym->bit_offset = bit_offset;
164                 sym->ctype.base_type->bit_offset = bit_offset;
165                 info->bit_size = bit_size + width;
166                 // warning (sym->pos, "bitfield: offset=%d:%d  size=:%d", sym->o

168                 return;
169         }

171         /*
172          * Otherwise, just align it right and add it up..
173          */
174         bit_size = (bit_size + align_bit_mask) & ~align_bit_mask;
175         sym->offset = bits_to_bytes(bit_size);

177         info->bit_size = bit_size + base_size;
178         // warning (sym->pos, "regular: offset=%d", sym->offset);
179 }

181 static struct symbol * examine_struct_union_type(struct symbol *sym, int advance
182 {
183         struct struct_union_info info = {
184                 .max_align = 1,
185                 .bit_size = 0,
186                 .align_size = 1
187         };
188         unsigned long bit_size, bit_align;
189         void (*fn)(struct symbol *, struct struct_union_info *);
190         struct symbol *member;

192         fn = advance ? lay_out_struct : lay_out_union;
```



```
193         FOR_EACH_PTR(sym->symbol_list, member) {
194                 fn(member, &info);
195         } END_FOR_EACH_PTR(member);

197         if (!sym->ctype.alignment)
198                 sym->ctype.alignment = info.max_align;
199         bit_size = info.bit_size;
200         if (info.align_size) {
201                 bit_align = bytes_to_bits(sym->ctype.alignment)-1;
202                 bit_size = (bit_size + bit_align) & ~bit_align;
203         }
204         sym->bit_size = bit_size;
205         return sym;
206 }

208 static struct symbol *examine_base_type(struct symbol *sym)
209 {
210         struct symbol *base_type;

212         /* Check the base type */
213         base_type = examine_symbol_type(sym->ctype.base_type);
214         if (!base_type || base_type->type == SYM_PTR)
215                 return base_type;
216         sym->ctype.as |= base_type->ctype.as;
217         sym->ctype.modifiers |= base_type->ctype.modifiers & MOD_PTRINHERIT;
218         concat_ptr_list((struct ptr_list *)base_type->ctype.contexts,
219                         (struct ptr_list **)&sym->ctype.contexts);
220         if (base_type->type == SYM_NODE) {
221                 base_type = base_type->ctype.base_type;
222                 sym->ctype.base_type = base_type;
223         }
224         return base_type;
225 }

227 static struct symbol * examine_array_type(struct symbol *sym)
228 {
229         struct symbol *base_type = examine_base_type(sym);
230         unsigned long bit_size = -1, alignment;
231         struct expression *array_size = sym->array_size;

233         if (!base_type)
234                 return sym;

236         if (array_size) {
237                 bit_size = array_element_offset(base_type->bit_size,
238                                                 get_expression_value_silent(arra
239                 if (array_size->type != EXPR_VALUE) {
240                         if (Wvla)
241                                 warning(array_size->pos, "Variable length array
242                         bit_size = -1;
243                 }
244         }
245         alignment = base_type->ctype.alignment;
246         if (!sym->ctype.alignment)
247                 sym->ctype.alignment = alignment;
248         sym->bit_size = bit_size;
249         return sym;
250 }

252 static struct symbol *examine_bitfield_type(struct symbol *sym)
253 {
254         struct symbol *base_type = examine_base_type(sym);
255         unsigned long bit_size, alignment, modifiers;

257         if (!base_type)
258                 return sym;
```

```
259         bit_size = base_type->bit_size;
260         if (sym->bit_size > bit_size)
261                 warning(sym->pos, "impossible field-width, %d, for this type",

263         alignment = base_type->ctype.alignment;
264         if (!sym->ctype.alignment)
265                 sym->ctype.alignment = alignment;
266         modifiers = base_type->ctype.modifiers;

268         /* Bitfields are unsigned, unless the base type was explicitly signed */
269         if (!(modifiers & MOD_EXPLICITLY_SIGNED))
270                 modifiers = (modifiers & ~MOD_SIGNED) | MOD_UNSIGNED;
271         sym->ctype.modifiers |= modifiers & MOD_SIGNEDNESS;
272         return sym;
273 }

275 /*
276  * "typeof" will have to merge the types together
277  */
278 void merge_type(struct symbol *sym, struct symbol *base_type)
279 {
280         sym->ctype.as |= base_type->ctype.as;
281         sym->ctype.modifiers |= (base_type->ctype.modifiers & ~MOD_STORAGE);
282         concat_ptr_list((struct ptr_list *)base_type->ctype.contexts,
283                         (struct ptr_list **)&sym->ctype.contexts);
284         sym->ctype.base_type = base_type->ctype.base_type;
285         if (sym->ctype.base_type->type == SYM_NODE)
286                 merge_type(sym, sym->ctype.base_type);
287 }

289 static int count_array_initializer(struct symbol *t, struct expression *expr)
290 {
291         int nr = 0;
292         int is_char = 0;

294         /*
295          * Arrays of character types are special; they can be initialized by
296          * string literal _or_ by string literal in braces.  The latter means
297          * that with T x[] = {<string literal>} number of elements in x depends
298          * on T - if it's a character type, we get the length of string literal
299          * (including NUL), otherwise we have one element here.
300          */
301         if (t->ctype.base_type == &int_type && t->ctype.modifiers & MOD_CHAR)
302                 is_char = 1;

304         switch (expr->type) {
305         case EXPR_INITIALIZER: {
306                 struct expression *entry;
307                 int count = 0;
308                 int str_len = 0;
309                 FOR_EACH_PTR(expr->expr_list, entry) {
310                         count++;
311                         switch (entry->type) {
312                         case EXPR_INDEX:
313                                 if (entry->idx_to >= nr)
314                                         nr = entry->idx_to+1;
315                                 break;
316                         case EXPR_PREOP: {
317                                 struct expression *e = entry;
318                                 if (is_char) {
319                                         while (e && e->type == EXPR_PREOP && e->
320                                                 e = e->unop;
321                                         if (e && e->type == EXPR_STRING) {
322                                                 entry = e;
323                         case EXPR_STRING:
324                                                 if (is_char)
```



```
325                                                         str_len = entry->string-
326                                         }

329                                 }
330                         }
331                         default:
332                                 nr++;
333                         }
334                 } END_FOR_EACH_PTR(entry);
335                 if (count == 1 && str_len)
336                         nr = str_len;
337                 break;
338         }
339         case EXPR_PREOP:
340                 if (is_char) {
341                         struct expression *e = expr;
342                         while (e && e->type == EXPR_PREOP && e->op == '(')
343                                 e = e->unop;
344                         if (e && e->type == EXPR_STRING) {
345                                 expr = e;
346         case EXPR_STRING:
347                                 if (is_char)
348                                         nr = expr->string->length;
349                         }
350                 }
351                 break;
352         default:
353                 break;
354         }
355         return nr;
356 }

358 static struct expression *get_symbol_initializer(struct symbol *sym)
359 {
360         do {
361                 if (sym->initializer)
362                         return sym->initializer;
363         } while ((sym = sym->same_symbol) != NULL);
364         return NULL;
365 }

367 static struct symbol * examine_node_type(struct symbol *sym)
368 {
369         struct symbol *base_type = examine_base_type(sym);
370         int bit_size;
371         unsigned long alignment;

373         /* SYM_NODE - figure out what the type of the node was.. */
374         bit_size = 0;
375         alignment = 0;
376         if (!base_type)
377                 return sym;

379         bit_size = base_type->bit_size;
380         alignment = base_type->ctype.alignment;

382         /* Pick up signedness information into the node */
383         sym->ctype.modifiers |= (MOD_SIGNEDNESS & base_type->ctype.modifiers);

385         if (!sym->ctype.alignment)
386                 sym->ctype.alignment = alignment;

388         /* Unsized array? The size might come from the initializer.. */
389         if (bit_size < 0 && base_type->type == SYM_ARRAY) {
390                 struct expression *initializer = get_symbol_initializer(sym);
```

```
391                 if (initializer) {
392                         struct symbol *node_type = base_type->ctype.base_type;
393                         int count = count_array_initializer(node_type, initializ
395                         if (node_type && node_type->bit_size >= 0)
396                                 bit_size = array_element_offset(node_type->bit_s
397                 }
398         }
399
400         sym->bit_size = bit_size;
401         return sym;
402 }
404 static struct symbol *examine_enum_type(struct symbol *sym)
405 {
406         struct symbol *base_type = examine_base_type(sym);
408         sym->ctype.modifiers |= (base_type->ctype.modifiers & MOD_SIGNEDNESS);
409         sym->bit_size = bits_in_enum;
410         if (base_type->bit_size > sym->bit_size)
411                 sym->bit_size = base_type->bit_size;
412         sym->ctype.alignment = enum_alignment;
413         if (base_type->ctype.alignment > sym->ctype.alignment)
414                 sym->ctype.alignment = base_type->ctype.alignment;
415         return sym;
416 }
418 static struct symbol *examine_pointer_type(struct symbol *sym)
419 {
420         /*
421          * We need to set the pointer size first, and
422          * examine the thing we point to only afterwards.
423          * That's because this pointer type may end up
424          * being needed for the base type size evaluation.
425          */
426         if (!sym->bit_size)
427                 sym->bit_size = bits_in_pointer;
428         if (!sym->ctype.alignment)
429                 sym->ctype.alignment = pointer_alignment;
430         return sym;
431 }
433 /*
434  * Fill in type size and alignment information for
435  * regular SYM_TYPE things.
436  */
437 struct symbol *examine_symbol_type(struct symbol * sym)
438 {
439         if (!sym)
440                 return sym;
442         /* Already done? */
443         if (sym->examined)
444                 return sym;
445         sym->examined = 1;
447         switch (sym->type) {
448         case SYM_FN:
449         case SYM_NODE:
450                 return examine_node_type(sym);
451         case SYM_ARRAY:
452                 return examine_array_type(sym);
453         case SYM_STRUCT:
454                 return examine_struct_union_type(sym, 1);
455         case SYM_UNION:
456                 return examine_struct_union_type(sym, 0);
```



```
457         case SYM_PTR:
458                 return examine_pointer_type(sym);
459         case SYM_ENUM:
460                 return examine_enum_type(sym);
461         case SYM_BITFIELD:
462                 return examine_bitfield_type(sym);
463         case SYM_BASETYPE:
464                 /* Size and alignment had better already be set up */
465                 return sym;
466         case SYM_TYPEOF: {
467                 struct symbol *base = evaluate_expression(sym->initializer);
468                 if (base) {
469                         unsigned long mod = 0;
471                         if (is_bitfield_type(base))
472                                 warning(base->pos, "typeof applied to bitfield t
473                         if (base->type == SYM_NODE) {
474                                 mod |= base->ctype.modifiers & MOD_TYPEOF;
475                                 base = base->ctype.base_type;
476                         }
477                         sym->type = SYM_NODE;
478                         sym->ctype.modifiers = mod;
479                         sym->ctype.base_type = base;
480                         return examine_node_type(sym);
481                 }
482                 break;
483         }
484         case SYM_PREPROCESSOR:
485                 sparse_error(sym->pos, "ctype on preprocessor command? (%s)", sh
486                 return NULL;
487         case SYM_UNINITIALIZED:
488 //              sparse_error(sym->pos, "ctype on uninitialized symbol %p", sym);
489                 return NULL;
490         case SYM_RESTRICT:
491                 examine_base_type(sym);
492                 return sym;
493         case SYM_FOULED:
494                 examine_base_type(sym);
495                 return sym;
496         default:
497 //              sparse_error(sym->pos, "Examining unknown symbol type %d", sym->
498                 break;
499         }
500         return sym;
501 }
503 const char* get_type_name(enum type type)
504 {
505         const char *type_lookup[] = {
506         [SYM_UNINITIALIZED] = "uninitialized",
507         [SYM_PREPROCESSOR] = "preprocessor",
508         [SYM_BASETYPE] = "basetype",
509         [SYM_NODE] = "node",
510         [SYM_PTR] = "pointer",
511         [SYM_FN] = "function",
512         [SYM_ARRAY] = "array",
513         [SYM_STRUCT] = "struct",
514         [SYM_UNION] = "union",
515         [SYM_ENUM] = "enum",
516         [SYM_TYPEDEF] = "typedef",
517         [SYM_TYPEOF] = "typeof",
518         [SYM_MEMBER] = "member",
519         [SYM_BITFIELD] = "bitfield",
520         [SYM_LABEL] = "label",
521         [SYM_RESTRICT] = "restrict",
522         [SYM_FOULED] = "fouled",
```

```
523         [SYM_KEYWORD] = "keyword",
524         [SYM_BAD] = "bad"};

526         if (type <= SYM_BAD)
527                 return type_lookup[type];
528         else
529                 return NULL;
530 }

532 struct symbol *examine_pointer_target(struct symbol *sym)
533 {
534         return examine_base_type(sym);
535 }

537 static struct symbol_list *restr, *fouled;

539 void create_fouled(struct symbol *type)
540 {
541         if (type->bit_size < bits_in_int) {
542                 struct symbol *new = alloc_symbol(type->pos, type->type);
543                 *new = *type;
544                 new->bit_size = bits_in_int;
545                 new->type = SYM_FOULED;
546                 new->ctype.base_type = type;
547                 add_symbol(&restr, type);
548                 add_symbol(&fouled, new);
549         }
550 }

552 struct symbol *befoul(struct symbol *type)
553 {
554         struct symbol *t1, *t2;
555         while (type->type == SYM_NODE)
556                 type = type->ctype.base_type;
557         PREPARE_PTR_LIST(restr, t1);
558         PREPARE_PTR_LIST(fouled, t2);
559         for (;;) {
560                 if (t1 == type)
561                         return t2;
562                 if (!t1)
563                         break;
564                 NEXT_PTR_LIST(t1);
565                 NEXT_PTR_LIST(t2);
566         }
567         FINISH_PTR_LIST(t2);
568         FINISH_PTR_LIST(t1);
569         return NULL;
570 }

572 void check_declaration(struct symbol *sym)
573 {
574         int warned = 0;
575         struct symbol *next = sym;

577         while ((next = next->next_id) != NULL) {
578                 if (next->namespace != sym->namespace)
579                         continue;
580                 if (sym->scope == next->scope) {
581                         sym->same_symbol = next;
582                         return;
583                 }
584                 /* Extern in block level matches a TOPLEVEL non-static symbol */
585                 if (sym->ctype.modifiers & MOD_EXTERN) {
586                         if ((next->ctype.modifiers & (MOD_TOPLEVEL|MOD_STATIC))
587                                 sym->same_symbol = next;
588                                 return;
```



```
589                         }
590                 }

592                 if (!Wshadow || warned)
593                         continue;
594                 if (get_sym_type(next) == SYM_FN)
595                         continue;
596                 warned = 1;
597                 warning(sym->pos, "symbol '%s' shadows an earlier one", show_ide
598                 info(next->pos, "originally declared here");
599         }
600 }

602 void bind_symbol(struct symbol *sym, struct ident *ident, enum namespace ns)
603 {
604         struct scope *scope;
605         if (sym->bound) {
606                 sparse_error(sym->pos, "internal error: symbol type already boun
607                 return;
608         }
609         if (ident->reserved && (ns & (NS_TYPEDEF | NS_STRUCT | NS_LABEL | NS_SYM
610                 sparse_error(sym->pos, "Trying to use reserved word '%s' as iden
611                 return;
612         }
613         sym->namespace = ns;
614         sym->next_id = ident->symbols;
615         ident->symbols = sym;
616         if (sym->ident && sym->ident != ident)
617                 warning(sym->pos, "Symbol '%s' already bound", show_ident(sym->i
618         sym->ident = ident;
619         sym->bound = 1;

621         scope = block_scope;
622         if (ns == NS_SYMBOL && toplevel(scope)) {
623                 unsigned mod = MOD_ADDRESSABLE | MOD_TOPLEVEL;

625                 scope = global_scope;
626                 if (sym->ctype.modifiers & MOD_STATIC ||
627                     is_extern_inline(sym)) {
628                         scope = file_scope;
629                         mod = MOD_TOPLEVEL;
630                 }
631                 sym->ctype.modifiers |= mod;
632         }
633         if (ns == NS_MACRO)
634                 scope = file_scope;
635         if (ns == NS_LABEL)
636                 scope = function_scope;
637         bind_scope(sym, scope);
638 }

640 struct symbol *create_symbol(int stream, const char *name, int type, int namespa
641 {
642         struct ident *ident = built_in_ident(name);
643         struct symbol *sym = lookup_symbol(ident, namespace);

645         if (sym && sym->type != type)
646                 die("symbol %s created with different types: %d old %d", name,
647                                 type, sym->type);

649         if (!sym) {
650                 struct token *token = built_in_token(stream, ident);

652                 sym = alloc_symbol(token->pos, type);
653                 bind_symbol(sym, token->ident, namespace);
654         }
```

```
655         return sym;
656 }


659 /*
660  * Abstract types
661  */
662 struct symbol   int_type,
663                 fp_type;

665 /*
666  * C types (i.e. actual instances that the abstract types
667  * can map onto)
668  */
669 struct symbol   bool_ctype, void_ctype, type_ctype,
670                 char_ctype, schar_ctype, uchar_ctype,
671                 short_ctype, sshort_ctype, ushort_ctype,
672                 int_ctype, sint_ctype, uint_ctype,
673                 long_ctype, slong_ctype, ulong_ctype,
674                 llong_ctype, sllong_ctype, ullong_ctype,
675                 lllong_ctype, slllong_ctype, ulllong_ctype,
676                 float_ctype, double_ctype, ldouble_ctype,
677                 string_ctype, ptr_ctype, lazy_ptr_ctype,
678                 incomplete_ctype, label_ctype, bad_ctype,
679                 null_ctype;

681 struct symbol   zero_int;

683 #define __INIT_IDENT(str, res) { .len = sizeof(str)-1, .name = str, .reserved =
684 #define __IDENT(n,str,res) \
685         struct ident n  = __INIT_IDENT(str,res)

687 #include "ident-list.h"

689 void init_symbols(void)
690 {
691         int stream = init_stream("builtin", -1, includepath);

693 #define __IDENT(n,str,res) \
694         hash_ident(&n)
695 #include "ident-list.h"

697         init_parser(stream);
698         init_builtins(stream);
699 }

701 #ifdef __CHAR_UNSIGNED__
702 #define CHAR_SIGNEDNESS MOD_UNSIGNED
703 #else
704 #define CHAR_SIGNEDNESS MOD_SIGNED
705 #endif

707 #define MOD_ESIGNED (MOD_SIGNED | MOD_EXPLICITLY_SIGNED)
708 #define MOD_LL (MOD_LONG | MOD_LONGLONG)
709 #define MOD_LLL MOD_LONGLONGLONG
710 static const struct ctype_declare {
711         struct symbol *ptr;
712         enum type type;
713         unsigned long modifiers;
714         int *bit_size;
715         int *maxalign;
716         struct symbol *base_type;
717 } ctype_declaration[] = {
718         { &bool_ctype,      SYM_BASETYPE, MOD_UNSIGNED,             &bits_in_boo
719         { &void_ctype,      SYM_BASETYPE, 0,                        NULL,
720         { &type_ctype,      SYM_BASETYPE, MOD_TYPE,                 NULL,
```



```
721         { &incomplete_ctype,SYM_BASETYPE, 0,                        NULL,
722         { &bad_ctype,       SYM_BASETYPE, 0,                        NULL,

724         { &char_ctype,      SYM_BASETYPE, CHAR_SIGNEDNESS | MOD_CHAR,    &bits_i
725         { &schar_ctype,     SYM_BASETYPE, MOD_ESIGNED | MOD_CHAR,   &bits_in_cha
726         { &uchar_ctype,     SYM_BASETYPE, MOD_UNSIGNED | MOD_CHAR,  &bits_in_cha
727         { &short_ctype,     SYM_BASETYPE, MOD_SIGNED | MOD_SHORT,   &bits_in_sho
728         { &sshort_ctype,    SYM_BASETYPE, MOD_ESIGNED | MOD_SHORT,  &bits_in_sho
729         { &ushort_ctype,    SYM_BASETYPE, MOD_UNSIGNED | MOD_SHORT, &bits_in_sho
730         { &int_ctype,       SYM_BASETYPE, MOD_SIGNED,               &bits_in_int
731         { &sint_ctype,      SYM_BASETYPE, MOD_ESIGNED,              &bits_in_int
732         { &uint_ctype,      SYM_BASETYPE, MOD_UNSIGNED,             &bits_in_int
733         { &long_ctype,      SYM_BASETYPE, MOD_SIGNED | MOD_LONG,    &bits_in_lon
734         { &slong_ctype,     SYM_BASETYPE, MOD_ESIGNED | MOD_LONG,   &bits_in_lon
735         { &ulong_ctype,     SYM_BASETYPE, MOD_UNSIGNED | MOD_LONG,  &bits_in_lon
736         { &llong_ctype,     SYM_BASETYPE, MOD_SIGNED | MOD_LL,      &bits_in_lon
737         { &sllong_ctype,    SYM_BASETYPE, MOD_ESIGNED | MOD_LL,     &bits_in_lon
738         { &ullong_ctype,    SYM_BASETYPE, MOD_UNSIGNED | MOD_LL,    &bits_in_lon
739         { &lllong_ctype,    SYM_BASETYPE, MOD_SIGNED | MOD_LLL,     &bits_in_lon
740         { &slllong_ctype,   SYM_BASETYPE, MOD_ESIGNED | MOD_LLL,    &bits_in_lon
741         { &ulllong_ctype,   SYM_BASETYPE, MOD_UNSIGNED | MOD_LLL,   &bits_in_lon

743         { &float_ctype,     SYM_BASETYPE,  0,                       &bits_in_flo
744         { &double_ctype,    SYM_BASETYPE, MOD_LONG,                 &bits_in_dou
745         { &ldouble_ctype,   SYM_BASETYPE, MOD_LONG | MOD_LONGLONG,  &bits_in_lon

747         { &string_ctype,    SYM_PTR,      0,                        &bits_in_poi
748         { &ptr_ctype,       SYM_PTR,      0,                        &bits_in_poi
749         { &null_ctype,      SYM_PTR,      0,                        &bits_in_poi
750         { &label_ctype,     SYM_PTR,      0,                        &bits_in_poi
751         { &lazy_ptr_ctype,  SYM_PTR,      0,                        &bits_in_poi
752         { NULL, }
753 };
754 #undef MOD_LLL
755 #undef MOD_LL
756 #undef MOD_ESIGNED

758 void init_ctype(void)
759 {
760         const struct ctype_declare *ctype;

762         for (ctype = ctype_declaration ; ctype->ptr; ctype++) {
763                 struct symbol *sym = ctype->ptr;
764                 unsigned long bit_size = ctype->bit_size ? *ctype->bit_size : -1
765                 unsigned long maxalign = ctype->maxalign ? *ctype->maxalign : 0;
766                 unsigned long alignment = bits_to_bytes(bit_size);

768                 if (alignment > maxalign)
769                         alignment = maxalign;
770                 sym->type = ctype->type;
771                 sym->bit_size = bit_size;
772                 sym->ctype.alignment = alignment;
773                 sym->ctype.base_type = ctype->base_type;
774                 sym->ctype.modifiers = ctype->modifiers;
775         }
776 }
```

```
**********************************************************
   12416 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/symbol.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef SYMBOL_H
#define SYMBOL_H
/*
 * Basic symbol and namespace definitions.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include "token.h"
#include "target.h"

/*
 * An identifier with semantic meaning is a "symbol".
 *
 * There's a 1:n relationship: each symbol is always
 * associated with one identifier, while each identifier
 * can have one or more semantic meanings due to C scope
 * rules.
 *
 * The progression is symbol -> token -> identifier. The
 * token contains the information on where the symbol was
 * declared.
 */
enum namespace {
        NS_NONE = 0,
        NS_MACRO = 1,
        NS_TYPEDEF = 2,
        NS_STRUCT = 4,  // Also used for unions and enums.
        NS_LABEL = 8,
        NS_SYMBOL = 16,
        NS_ITERATOR = 32,
        NS_PREPROCESSOR = 64,
        NS_UNDEF = 128,
        NS_KEYWORD = 256,
};

enum type {
        SYM_UNINITIALIZED,
        SYM_PREPROCESSOR,
        SYM_BASETYPE,
        SYM_NODE,
```



```
        SYM_PTR,
        SYM_FN,
        SYM_ARRAY,
        SYM_STRUCT,
        SYM_UNION,
        SYM_ENUM,
        SYM_TYPEDEF,
        SYM_TYPEOF,
        SYM_MEMBER,
        SYM_BITFIELD,
        SYM_LABEL,
        SYM_RESTRICT,
        SYM_FOULED,
        SYM_KEYWORD,
        SYM_BAD,
};

enum keyword {
        KW_SPECIFIER    = 1 << 0,
        KW_MODIFIER     = 1 << 1,
        KW_QUALIFIER    = 1 << 2,
        KW_ATTRIBUTE    = 1 << 3,
        KW_STATEMENT    = 1 << 4,
        KW_ASM          = 1 << 5,
        KW_MODE         = 1 << 6,
        KW_SHORT        = 1 << 7,
        KW_LONG         = 1 << 8,
        KW_EXACT        = 1 << 9,
};

struct context {
        struct expression *context;
        unsigned int in, out;
};

extern struct context *alloc_context(void);

DECLARE_PTR_LIST(context_list, struct context);

struct ctype {
        unsigned long modifiers;
        unsigned long alignment;
        struct context_list *contexts;
        unsigned int as;
        struct symbol *base_type;
};

struct decl_state {
        struct ctype ctype;
        struct ident **ident;
        struct symbol_op *mode;
        unsigned char prefer_abstract, is_inline, storage_class, is_tls;
};

struct symbol_op {
        enum keyword type;
        int (*evaluate)(struct expression *);
        int (*expand)(struct expression *, int);
        int (*args)(struct expression *);

        /* keywords */
        struct token *(*declarator)(struct token *token, struct decl_state *ctx)
        struct token *(*statement)(struct token *token, struct statement *stmt);
        struct token *(*toplevel)(struct token *token, struct symbol_list **list
        struct token *(*attribute)(struct token *token, struct symbol *attr, str
        struct symbol *(*to_mode)(struct symbol *);
```

```
128         int test, set, class;
129 };


132 #define SYM_ATTR_WEAK           0
133 #define SYM_ATTR_NORMAL         1
134 #define SYM_ATTR_STRONG         2

136 struct symbol {
137         enum type type:8;
138         enum namespace namespace:9;
139         unsigned char used:1, attr:2, enum_member:1, bound:1;
140         struct position pos;            /* Where this symbol was declared */
141         struct position endpos;         /* Where this symbol ends*/
142         struct ident *ident;            /* What identifier this symbol is associ
143         struct symbol *next_id;         /* Next semantic symbol that shares this
144         struct symbol   *replace;       /* What is this symbol shadowed by in co
145         struct scope    *scope;
146         union {
147                 struct symbol   *same_symbol;
148                 struct symbol   *next_subobject;
149         };

151         struct symbol_op *op;

153         union {
154                 struct /* NS_MACRO */ {
155                         struct token *expansion;
156                         struct token *arglist;
157                         struct scope *used_in;
158                 };
159                 struct /* NS_PREPROCESSOR */ {
160                         int (*handler)(struct stream *, struct token **, struct
161                         int normal;
162                 };
163                 struct /* NS_SYMBOL */ {
164                         unsigned long   offset;
165                         int             bit_size;
166                         unsigned int    bit_offset:8,
167                                         arg_count:10,
168                                         variadic:1,
169                                         initialized:1,
170                                         examined:1,
171                                         expanding:1,
172                                         evaluated:1,
173                                         string:1,
174                                         designated_init:1,
175                                         forced_arg:1,
176                                         transparent_union:1;
177                         struct expression *array_size;
178                         struct ctype ctype;
179                         struct symbol_list *arguments;
180                         struct statement *stmt;
181                         struct symbol_list *symbol_list;
182                         struct statement *inline_stmt;
183                         struct symbol_list *inline_symbol_list;
184                         struct expression *initializer;
185                         struct entrypoint *ep;
186                         long long value;                /* Initial value */
187                         struct symbol *definition;
188                 };
189         };
190         union /* backend */ {
191                 struct basic_block *bb_target;  /* label */
192                 void *aux;                      /* Auxiliary info, e.g. backend
```



```
193                 struct {                        /* sparse ctags */
194                         char kind;
195                         unsigned char visited:1;
196                 };
197         };
198         pseudo_t pseudo;
199 };

201 /* Modifiers */
202 #define MOD_AUTO        0x0001
203 #define MOD_REGISTER    0x0002
204 #define MOD_STATIC      0x0004
205 #define MOD_EXTERN      0x0008

207 #define MOD_CONST       0x0010
208 #define MOD_VOLATILE    0x0020
209 #define MOD_SIGNED      0x0040
210 #define MOD_UNSIGNED    0x0080

212 #define MOD_CHAR        0x0100
213 #define MOD_SHORT       0x0200
214 #define MOD_LONG        0x0400
215 #define MOD_LONGLONG    0x0800
216 #define MOD_LONGLONGLONG        0x1000
217 #define MOD_PURE        0x2000

219 #define MOD_TYPEDEF     0x10000

221 #define MOD_TLS         0x20000
222 #define MOD_INLINE      0x40000
223 #define MOD_ADDRESSABLE 0x80000

225 #define MOD_NOCAST      0x100000
226 #define MOD_NODEREF     0x200000
227 #define MOD_ACCESSED    0x400000
228 #define MOD_TOPLEVEL    0x800000        // scoping..

230 #define MOD_ASSIGNED    0x2000000
231 #define MOD_TYPE        0x4000000
232 #define MOD_SAFE        0x8000000       // non-null/non-trapping pointer

234 #define MOD_USERTYPE    0x10000000
235 #define MOD_NORETURN    0x20000000
236 #define MOD_EXPLICITLY_SIGNED   0x40000000
237 #define MOD_BITWISE     0x80000000


240 #define MOD_NONLOCAL    (MOD_EXTERN | MOD_TOPLEVEL)
241 #define MOD_STORAGE     (MOD_AUTO | MOD_REGISTER | MOD_STATIC | MOD_EXTERN | MOD
242 #define MOD_SIGNEDNESS  (MOD_SIGNED | MOD_UNSIGNED | MOD_EXPLICITLY_SIGNED)
243 #define MOD_LONG_ALL    (MOD_LONG | MOD_LONGLONG | MOD_LONGLONGLONG)
244 #define MOD_SPECIFIER   (MOD_CHAR | MOD_SHORT | MOD_LONG_ALL | MOD_SIGNEDNESS)
245 #define MOD_SIZE        (MOD_CHAR | MOD_SHORT | MOD_LONG_ALL)
246 #define MOD_IGNORE (MOD_TOPLEVEL | MOD_STORAGE | MOD_ADDRESSABLE |      \
247         MOD_ASSIGNED | MOD_USERTYPE | MOD_ACCESSED | MOD_EXPLICITLY_SIGNED)
248 #define MOD_PTRINHERIT (MOD_VOLATILE | MOD_CONST | MOD_NODEREF | MOD_NORETURN |
249 /* modifiers preserved by typeof() operator */
250 #define MOD_TYPEOF      (MOD_VOLATILE | MOD_CONST | MOD_NOCAST | MOD_SPECIFIER)


253 /* Current parsing/evaluation function */
254 extern struct symbol *current_fn;

256 /* Abstract types */
257 extern struct symbol    int_type,
258                         fp_type;
```

```
260 /* C types */
261 extern struct symbol    bool_ctype, void_ctype, type_ctype,
262                         char_ctype, schar_ctype, uchar_ctype,
263                         short_ctype, sshort_ctype, ushort_ctype,
264                         int_ctype, sint_ctype, uint_ctype,
265                         long_ctype, slong_ctype, ulong_ctype,
266                         llong_ctype, sllong_ctype, ullong_ctype,
267                         lllong_ctype, slllong_ctype, ulllong_ctype,
268                         float_ctype, double_ctype, ldouble_ctype,
269                         string_ctype, ptr_ctype, lazy_ptr_ctype,
270                         incomplete_ctype, label_ctype, bad_ctype,
271                         null_ctype;

273 /* Special internal symbols */
274 extern struct symbol    zero_int;

276 #define __IDENT(n,str,res) \
277         extern struct ident n
278 #include "ident-list.h"

280 #define symbol_is_typename(sym) ((sym)->type == SYM_TYPE)

282 extern struct symbol_list *translation_unit_used_list;

284 extern void access_symbol(struct symbol *);

286 extern const char * type_difference(struct ctype *c1, struct ctype *c2,
287         unsigned long mod1, unsigned long mod2);

289 extern struct symbol *lookup_symbol(struct ident *, enum namespace);
290 extern struct symbol *create_symbol(int stream, const char *name, int type, int
291 extern void init_symbols(void);
292 extern void init_builtins(int stream);
293 extern void init_ctype(void);
294 extern struct symbol *alloc_symbol(struct position, int type);
295 extern void show_type(struct symbol *);
296 extern const char *modifier_string(unsigned long mod);
297 extern void show_symbol(struct symbol *);
298 extern int show_symbol_expr_init(struct symbol *sym);
299 extern void show_type_list(struct symbol *);
300 extern void show_symbol_list(struct symbol_list *, const char *);
301 extern void add_symbol(struct symbol_list **, struct symbol *);
302 extern void bind_symbol(struct symbol *, struct ident *, enum namespace);

304 extern struct symbol *examine_symbol_type(struct symbol *);
305 extern struct symbol *examine_pointer_target(struct symbol *);
306 extern void examine_simple_symbol_type(struct symbol *);
307 extern const char *show_typename(struct symbol *sym);
308 extern const char *builtin_typename(struct symbol *sym);
309 extern const char *builtin_ctypename(struct ctype *ctype);
310 extern const char* get_type_name(enum type type);

312 extern void debug_symbol(struct symbol *);
313 extern void merge_type(struct symbol *sym, struct symbol *base_type);
314 extern void check_declaration(struct symbol *sym);

316 static inline struct symbol *get_base_type(const struct symbol *sym)
317 {
318         return examine_symbol_type(sym->ctype.base_type);
319 }

321 static inline int is_int_type(const struct symbol *type)
322 {
323         if (type->type == SYM_NODE)
324                 type = type->ctype.base_type;
```



```
325         if (type->type == SYM_ENUM)
326                 type = type->ctype.base_type;
327         return type->type == SYM_BITFIELD ||
328                type->ctype.base_type == &int_type;
329 }

331 static inline int is_enum_type(const struct symbol *type)
332 {
333         if (type->type == SYM_NODE)
334                 type = type->ctype.base_type;
335         return (type->type == SYM_ENUM);
336 }

338 static inline int is_type_type(struct symbol *type)
339 {
340         return (type->ctype.modifiers & MOD_TYPE) != 0;
341 }

343 static inline int is_ptr_type(struct symbol *type)
344 {
345         if (!type)
346                 return 0;
347         if (type->type == SYM_NODE)
348                 type = type->ctype.base_type;
349         return type->type == SYM_PTR || type->type == SYM_ARRAY || type->type ==
350 }

352 static inline int is_func_type(struct symbol *type)
353 {
354         if (type->type == SYM_NODE)
355                 type = type->ctype.base_type;
356         return type->type == SYM_FN;
357 }

359 static inline int is_array_type(struct symbol *type)
360 {
361         if (type->type == SYM_NODE)
362                 type = type->ctype.base_type;
363         return type->type == SYM_ARRAY;
364 }

366 static inline int is_float_type(struct symbol *type)
367 {
368         if (type->type == SYM_NODE)
369                 type = type->ctype.base_type;
370         return type->ctype.base_type == &fp_type;
371 }

373 static inline int is_byte_type(struct symbol *type)
374 {
375         return type->bit_size == bits_in_char && type->type != SYM_BITFIELD;
376 }

378 static inline int is_void_type(struct symbol *type)
379 {
380         if (type->type == SYM_NODE)
381                 type = type->ctype.base_type;
382         return type == &void_ctype;
383 }

385 static inline int is_bool_type(struct symbol *type)
386 {
387         if (type->type == SYM_NODE)
388                 type = type->ctype.base_type;
389         return type == &bool_ctype;
390 }
```

```
392 static inline int is_scalar_type(struct symbol *type)
393 {
394         if (type->type == SYM_NODE)
395                 type = type->ctype.base_type;
396         switch (type->type) {
397         case SYM_ENUM:
398         case SYM_BITFIELD:
399         case SYM_PTR:
400         case SYM_RESTRICT:      // OK, always integer types
401                 return 1;
402         default:
403                 break;
404         }
405         if (type->ctype.base_type == &int_type)
406                 return 1;
407         if (type->ctype.base_type == &fp_type)
408                 return 1;
409         return 0;
410 }

412 static inline int is_function(struct symbol *type)
413 {
414         return type && type->type == SYM_FN;
415 }

417 static inline int is_extern_inline(struct symbol *sym)
418 {
419         return (sym->ctype.modifiers & MOD_EXTERN) &&
420                 (sym->ctype.modifiers & MOD_INLINE) &&
421                 is_function(sym->ctype.base_type);
422 }

424 static inline int get_sym_type(struct symbol *type)
425 {
426         if (type->type == SYM_NODE)
427                 type = type->ctype.base_type;
428         if (type->type == SYM_ENUM)
429                 type = type->ctype.base_type;
430         return type->type;
431 }

433 static inline struct symbol *lookup_keyword(struct ident *ident, enum namespace
434 {
435         if (!ident->keyword)
436                 return NULL;
437         return lookup_symbol(ident, ns);
438 }

440 #define is_restricted_type(type) (get_sym_type(type) == SYM_RESTRICT)
441 #define is_fouled_type(type) (get_sym_type(type) == SYM_FOULED)
442 #define is_bitfield_type(type)   (get_sym_type(type) == SYM_BITFIELD)
443 extern int is_ptr_type(struct symbol *);

445 void create_fouled(struct symbol *type);
446 struct symbol *befoul(struct symbol *type);

448 #endif /* SYMBOL_H */
```

```
**********************************************************
     768 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/target.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>

#include "symbol.h"
#include "target.h"

struct symbol *size_t_ctype = &ulong_ctype;
struct symbol *ssize_t_ctype = &long_ctype;

/*
 * For "__attribute__((aligned))"
 */
int max_alignment = 16;

/*
 * Integer data types
 */
int bits_in_bool = 1;
int bits_in_char = 8;
int bits_in_short = 16;
int bits_in_int = 32;
int bits_in_long = 32;
int bits_in_longlong = 64;
int bits_in_longlonglong = 128;

int bits_in_wchar = 32;

int max_int_alignment = 4;

/*
 * Floating point data types
 */
int bits_in_float = 32;
int bits_in_double = 64;
int bits_in_longdouble = 80;

int max_fp_alignment = 8;

/*
 * Pointer data type
 */
int bits_in_pointer = 32;
int pointer_alignment = 4;

/*
 * Enum data types
 */
int bits_in_enum = 32;
int enum_alignment = 4;
```

```
**********************************************************
    1251 Fri Dec 21 15:00:38 2018
new/usr/src/tools/smatch/src/target.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef TARGET_H
#define TARGET_H

extern struct symbol *size_t_ctype;
extern struct symbol *ssize_t_ctype;

/*
 * For "__attribute__((aligned))"
 */
extern int max_alignment;

/*
 * Integer data types
 */
extern int bits_in_bool;
extern int bits_in_char;
extern int bits_in_short;
extern int bits_in_int;
extern int bits_in_long;
extern int bits_in_longlong;
extern int bits_in_longlonglong;

extern int bits_in_wchar;

extern int max_int_alignment;

/*
 * Floating point data types
 */
extern int bits_in_float;
extern int bits_in_double;
extern int bits_in_longdouble;

extern int max_fp_alignment;

/*
 * Pointer data type
 */
extern int bits_in_pointer;
extern int pointer_alignment;

/*
 * Enum data types
 */
extern int bits_in_enum;
extern int enum_alignment;

/*
 * Helper functions for converting bits to bytes and vice versa.
 */

static inline int bits_to_bytes(int bits)
{
        return bits >= 0 ? (bits + bits_in_char - 1) / bits_in_char : -1;
}

static inline int bytes_to_bits(int bytes)
{
        return bytes * bits_in_char;
}
```



```
static inline unsigned long array_element_offset(unsigned long base_bits, int id
{
        int fragment = base_bits % bits_in_char;
        if (fragment)
                base_bits += bits_in_char - fragment;
        return base_bits * idx;
}

#endif
```

```
**********************************************************
    2137 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/test-dissect.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "dissect.h"

static unsigned dotc_stream;

static inline char storage(struct symbol *sym)
{
        int t = sym->type;
        unsigned m = sym->ctype.modifiers;

        if (m & MOD_INLINE || t == SYM_STRUCT || t == SYM_UNION /*|| t == SYM_EN
                return sym->pos.stream == dotc_stream ? 's' : 'g';

        return (m & MOD_STATIC) ? 's' : (m & MOD_NONLOCAL) ? 'g' : 'l';
}

static inline const char *show_mode(unsigned mode)
{
        static char str[3];

        if (mode == -1)
                return "def";

#define U(u_r)  "-rwm"[(mode / u_r) & 3]
        str[0] = U(U_R_AOF);
        str[1] = U(U_R_VAL);
        str[2] = U(U_R_PTR);
#undef  U

        return str;
}

static void print_usage(struct position *pos, struct symbol *sym, unsigned mode)
{
        static unsigned curr_stream = -1;

        if (curr_stream != pos->stream) {
                curr_stream = pos->stream;
                printf("\nFILE: %s\n\n", stream_name(curr_stream));
        }

        printf("%4d:%-3d %c %-5.3s",
                pos->line, pos->pos, storage(sym), show_mode(mode));
}

static void r_symbol(unsigned mode, struct position *pos, struct symbol *sym)
{
        print_usage(pos, sym, mode);

        if (!sym->ident)
                sym->ident = built_in_ident("__asm__");

        printf("%-32.*s %s\n",
                sym->ident->len, sym->ident->name,
                show_typename(sym->ctype.base_type));
}

static void r_member(unsigned mode, struct position *pos, struct symbol *sym, st
{
        struct ident *ni, *si, *mi;
```



```
        print_usage(pos, sym, mode);

        ni = built_in_ident("?");
        si = sym->ident ?: ni;
        /* mem == NULL means entire struct accessed */
        mi = mem ? (mem->ident ?: ni) : built_in_ident("*");

        printf("%.*s.%-*.*s %s\n",
                si->len, si->name,
                32-1 - si->len, mi->len, mi->name,
                show_typename(mem ? mem->ctype.base_type : sym));
}

static void r_symdef(struct symbol *sym)
{
        r_symbol(-1, &sym->pos, sym);
}

int main(int argc, char **argv)
{
        static struct reporter reporter = {
                .r_symdef = r_symdef,
                .r_symbol = r_symbol,
                .r_member = r_member,
        };
        struct string_list *filelist = NULL;
        char *file;

        sparse_initialize(argc, argv, &filelist);

        FOR_EACH_PTR_NOTAG(filelist, file) {
                dotc_stream = input_stream_nr;
                dissect(__sparse(file), &reporter);
        } END_FOR_EACH_PTR_NOTAG(file);

        return 0;
}
```

```
**********************************************************
     866 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/test-inspect.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"

#include "ast-view.h"

static void expand_symbols(struct symbol_list *list)
{
        struct symbol *sym;
        FOR_EACH_PTR(list, sym) {
                expand_symbol(sym);
        } END_FOR_EACH_PTR(sym);
}

int main(int argc, char **argv)
{
        struct string_list *filelist = NULL;
        char *file;
        struct symbol_list *view_syms = NULL;

        gtk_init(&argc,&argv);
        expand_symbols(sparse_initialize(argc, argv, &filelist));
        FOR_EACH_PTR_NOTAG(filelist, file) {
                struct symbol_list *syms = sparse(file);
                expand_symbols(syms);
                concat_symbol_list(syms, &view_syms);
        } END_FOR_EACH_PTR_NOTAG(file);
        treeview_main(view_syms);
        return 0;
}

```

```
**********************************************************
    1728 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/test-lexing.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Example test program that just uses the tokenization and
 * preprocessing phases, and prints out the results.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "token.h"
#include "symbol.h"

int main(int argc, char **argv)
{
        struct string_list *filelist = NULL;
        char *file;

        preprocess_only = 1;
        sparse_initialize(argc, argv, &filelist);
        FOR_EACH_PTR_NOTAG(filelist, file) {
                sparse(file);
        } END_FOR_EACH_PTR_NOTAG(file);
        show_identifier_stats();
        return 0;
}
```

```
**********************************************************
    2018 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/test-linearize.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Parse and linearize the tree for testing.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003-2004 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"
#include "linearize.h"

static void clean_up_symbols(struct symbol_list *list)
{
        struct symbol *sym;

        FOR_EACH_PTR(list, sym) {
                struct entrypoint *ep;

                expand_symbol(sym);
                ep = linearize_symbol(sym);
                if (ep)
                        show_entry(ep);
        } END_FOR_EACH_PTR(sym);
}

int main(int argc, char **argv)
{
        struct string_list *filelist = NULL;
        char *file;

        clean_up_symbols(sparse_initialize(argc, argv, &filelist));
```



```
        FOR_EACH_PTR_NOTAG(filelist, file) {
                clean_up_symbols(sparse(file));
        } END_FOR_EACH_PTR_NOTAG(file);

        report_stats();
        return 0;
}
```

```
**********************************************************
    2450 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/test-parsing.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Example trivial client program that uses the sparse library
 * to tokenize, preprocess and parse a C file, and prints out
 * the results.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "parse.h"
#include "symbol.h"
#include "expression.h"

static void clean_up_symbols(struct symbol_list *list)
{
        struct symbol *sym;

        FOR_EACH_PTR(list, sym) {
                expand_symbol(sym);
        } END_FOR_EACH_PTR(sym);
}

int main(int argc, char **argv)
{
        struct symbol_list * list;
        struct string_list * filelist = NULL;
        char *file;

        list = sparse_initialize(argc, argv, &filelist);

        // Simplification
        clean_up_symbols(list);
```



```
#if 1
        show_symbol_list(list, "\n\n");
        printf("\n\n");
#endif

        FOR_EACH_PTR_NOTAG(filelist, file) {
                list = sparse(file);

                // Simplification
                clean_up_symbols(list);

#if 1
                // Show the end result.
                show_symbol_list(list, "\n\n");
                printf("\n\n");
#endif
        } END_FOR_EACH_PTR_NOTAG(file);

#if 0
        // And show the allocation statistics
        show_ident_alloc();
        show_token_alloc();
        show_symbol_alloc();
        show_expression_alloc();
        show_statement_alloc();
        show_string_alloc();
        show_bytes_alloc();
#endif
        return 0;
}
```

```
**********************************************************
     852 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/test-sort.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "lib.h"
#include "allocate.h"
#include <stdio.h>
#include <stdlib.h>

static int
int_cmp (const void *_a, const void *_b)
{
  const int *a = _a;
  const int *b = _b;
  return *a - *b;
}

#define MIN(_x,_y) ((_x) < (_y) ? (_x) : (_y))

int
main (int argc, char **argv)
{
  struct ptr_list *l = NULL, *l2;
  int i, *e;
  const int N = argv[1] ? atoi (argv[1]) : 10000;

  srand (N);
  for (i = 0; i < 1000; i++)
    (void)rand ();

  for (i = 0; i < N; i++) {
    e = (int *)malloc (sizeof (int));
    *e = rand ();
    add_ptr_list (&l, e);
  }
  sort_list (&l, int_cmp);
  // Sort already sorted stuff.
  sort_list (&l, int_cmp);

  l2 = l;
  do {
    l2->nr = MIN (l2->nr, rand () % 3);
    for (i = 0; i < l2->nr; i++)
      *((int *)(l2->list[i])) = rand();
    l2 = l2->next;
  } while (l2 != l);
  sort_list (&l, int_cmp);

  return 0;
}
```

```
**********************************************************
    1654 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/test-unssa.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <assert.h>

#include "symbol.h"
#include "expression.h"
#include "linearize.h"
#include "flow.h"


static void output_bb(struct basic_block *bb, unsigned long generation)
{
        struct instruction *insn;

        bb->generation = generation;
        printf(".L%u\n", bb->nr);

        FOR_EACH_PTR(bb->insns, insn) {
                if (!insn->bb)
                        continue;
                printf("\t%s\n", show_instruction(insn));
        }
        END_FOR_EACH_PTR(insn);

        printf("\n");
}

static void output_fn(struct entrypoint *ep)
{
        struct basic_block *bb;
        unsigned long generation = ++bb_generation;
        struct symbol *sym = ep->name;
        const char *name = show_ident(sym->ident);

        if (sym->ctype.modifiers & MOD_STATIC)
                printf("\n\n%s:\n", name);
        else
                printf("\n\n.globl %s\n%s:\n", name, name);

        unssa(ep);

        FOR_EACH_PTR(ep->bbs, bb) {
                if (bb->generation == generation)
                        continue;
                output_bb(bb, generation);
        }
        END_FOR_EACH_PTR(bb);
}

static int output_data(struct symbol *sym)
{
        printf("symbol %s:\n", show_ident(sym->ident));
        printf("\ttype = %d\n", sym->ctype.base_type->type);
        printf("\tmodif= %lx\n", sym->ctype.modifiers);

        return 0;
}

static int compile(struct symbol_list *list)
{
        struct symbol *sym;
```



```
        FOR_EACH_PTR(list, sym) {
                struct entrypoint *ep;
                expand_symbol(sym);
                ep = linearize_symbol(sym);
                if (ep)
                        output_fn(ep);
                else
                        output_data(sym);
        }
        END_FOR_EACH_PTR(sym);

        return 0;
}

int main(int argc, char **argv)
{
        struct string_list * filelist = NULL;
        char *file;

        compile(sparse_initialize(argc, argv, &filelist));
        FOR_EACH_PTR_NOTAG(filelist, file) {
                compile(sparse(file));
        } END_FOR_EACH_PTR_NOTAG(file);

        report_stats();
        return 0;
}
```

```
**********************************************************
    6668 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/token.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef TOKEN_H
#define TOKEN_H
/*
 * Basic tokenization structures. NOTE! Those tokens had better
 * be pretty small, since we're going to keep them all in memory
 * indefinitely.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include <sys/types.h>
#include "lib.h"

/*
 * This describes the pure lexical elements (tokens), with
 * no semantic meaning. In other words, an identifier doesn't
 * have a type or meaning, it is only a specific string in
 * the input stream.
 *
 * Semantic meaning is handled elsewhere.
 */

enum constantfile {
  CONSTANT_FILE_MAYBE,    // To be determined, not inside any #ifs in this file
  CONSTANT_FILE_IFNDEF,   // To be determined, currently inside #ifndef
  CONSTANT_FILE_NOPE,     // No
  CONSTANT_FILE_YES       // Yes
};

extern const char *includepath[];

struct stream {
        int fd;
        const char *name;
        const char *path;    // input-file path - see set_stream_include_path()
        const char **next_path;

        /* Use these to check for "already parsed" */
        enum constantfile constant;
        int dirty, next_stream, once;
        struct ident *protect;
```



```
        struct token *ifndef;
        struct token *top_if;
};

extern int input_stream_nr;
extern struct stream *input_streams;
extern unsigned int tabstop;
extern int no_lineno;
extern int *hash_stream(const char *name);

struct ident {
        struct ident *next;     /* Hash chain of identifiers */
        struct symbol *symbols; /* Pointer to semantic meaning list */
        unsigned char len;      /* Length of identifier name */
        unsigned char tainted:1,
                      reserved:1,
                      keyword:1;
        char name[];            /* Actual identifier */
};

enum token_type {
        TOKEN_EOF,
        TOKEN_ERROR,
        TOKEN_IDENT,
        TOKEN_ZERO_IDENT,
        TOKEN_NUMBER,
        TOKEN_CHAR,
        TOKEN_CHAR_EMBEDDED_0,
        TOKEN_CHAR_EMBEDDED_1,
        TOKEN_CHAR_EMBEDDED_2,
        TOKEN_CHAR_EMBEDDED_3,
        TOKEN_WIDE_CHAR,
        TOKEN_WIDE_CHAR_EMBEDDED_0,
        TOKEN_WIDE_CHAR_EMBEDDED_1,
        TOKEN_WIDE_CHAR_EMBEDDED_2,
        TOKEN_WIDE_CHAR_EMBEDDED_3,
        TOKEN_STRING,
        TOKEN_WIDE_STRING,
        TOKEN_SPECIAL,
        TOKEN_STREAMBEGIN,
        TOKEN_STREAMEND,
        TOKEN_MACRO_ARGUMENT,
        TOKEN_STR_ARGUMENT,
        TOKEN_QUOTED_ARGUMENT,
        TOKEN_CONCAT,
        TOKEN_GNU_KLUDGE,
        TOKEN_UNTAINT,
        TOKEN_ARG_COUNT,
        TOKEN_IF,
        TOKEN_SKIP_GROUPS,
        TOKEN_ELSE,
};

/* Combination tokens */
#define COMBINATION_STRINGS {   \
        "+=", "++",             \
        "-=", "--", "->",       \
        "*=",                   \
        "/=",                   \
        "%=",                   \
        "<=", ">=",             \
        "==", "!=",             \
        "&&", "&=",             \
        "||", "|=",             \
        "^=", "##",             \
        "<<", ">>", "..",       \
```

```
127         "<<=", ">>=", "...",    \
128         "",                     \
129         "<", ">", "<=", ">="    \
130 }

132 extern unsigned char combinations[][4];

134 enum special_token {
135         SPECIAL_BASE = 256,
136         SPECIAL_ADD_ASSIGN = SPECIAL_BASE,
137         SPECIAL_INCREMENT,
138         SPECIAL_SUB_ASSIGN,
139         SPECIAL_DECREMENT,
140         SPECIAL_DEREFERENCE,
141         SPECIAL_MUL_ASSIGN,
142         SPECIAL_DIV_ASSIGN,
143         SPECIAL_MOD_ASSIGN,
144         SPECIAL_LTE,
145         SPECIAL_GTE,
146         SPECIAL_EQUAL,
147         SPECIAL_NOTEQUAL,
148         SPECIAL_LOGICAL_AND,
149         SPECIAL_AND_ASSIGN,
150         SPECIAL_LOGICAL_OR,
151         SPECIAL_OR_ASSIGN,
152         SPECIAL_XOR_ASSIGN,
153         SPECIAL_HASHHASH,
154         SPECIAL_LEFTSHIFT,
155         SPECIAL_RIGHTSHIFT,
156         SPECIAL_DOTDOT,
157         SPECIAL_SHL_ASSIGN,
158         SPECIAL_SHR_ASSIGN,
159         SPECIAL_ELLIPSIS,
160         SPECIAL_ARG_SEPARATOR,
161         SPECIAL_UNSIGNED_LT,
162         SPECIAL_UNSIGNED_GT,
163         SPECIAL_UNSIGNED_LTE,
164         SPECIAL_UNSIGNED_GTE,
165 };

167 struct string {
168         unsigned int length:31;
169         unsigned int immutable:1;
170         char data[];
171 };

173 /* will fit into 32 bits */
174 struct argcount {
175         unsigned normal:10;
176         unsigned quoted:10;
177         unsigned str:10;
178         unsigned vararg:1;
179 };

181 /*
182  * This is a very common data structure, it should be kept
183  * as small as humanly possible. Big (rare) types go as
184  * pointers.
185  */
186 struct token {
187         struct position pos;
188         struct token *next;
189         union {
190                 const char *number;
191                 struct ident *ident;
192                 unsigned int special;
```



```
193                 struct string *string;
194                 int argnum;
195                 struct argcount count;
196                 char embedded[4];
197         };
198 };

200 #define MAX_STRING 8191

202 static inline struct token *containing_token(struct token **p)
203 {
204         void *addr = (char *)p - ((char *)&((struct token *)0)->next - (char *)0
205         return addr;
206 }

208 #define token_type(x) ((x)->pos.type)

210 /*
211  * Last token in the stream - points to itself.
212  * This allows us to not test for NULL pointers
213  * when following the token->next chain..
214  */
215 extern struct token eof_token_entry;
216 #define eof_token(x) ((x) == &eof_token_entry)

218 extern int init_stream(const char *, int fd, const char **next_path);
219 extern const char *stream_name(int stream);
220 extern struct ident *hash_ident(struct ident *);
221 extern struct ident *built_in_ident(const char *);
222 extern struct token *built_in_token(int, struct ident *);
223 extern const char *show_special(int);
224 extern const char *show_ident(const struct ident *);
225 extern const char *show_string(const struct string *string);
226 extern const char *show_token(const struct token *);
227 extern const char *quote_token(const struct token *);
228 extern struct token * tokenize(const char *, int, struct token *, const char **n
229 extern struct token * tokenize_buffer(void *, unsigned long, struct token **);

231 extern void show_identifier_stats(void);
232 extern void init_include_path(void);
233 extern struct token *preprocess(struct token *);

235 extern void store_all_tokens(struct token *token);
236 extern struct token *pos_get_token(struct position pos);
237 extern char *pos_ident(struct position pos);

239 extern void store_macro_pos(struct token *);
240 extern char *get_macro_name(struct position pos);

242 static inline int match_op(struct token *token, unsigned int op)
243 {
244         return token->pos.type == TOKEN_SPECIAL && token->special == op;
245 }

247 static inline int match_ident(struct token *token, struct ident *id)
248 {
249         return token->pos.type == TOKEN_IDENT && token->ident == id;
250 }

252 #endif
```

```
**********************************************************
    3850 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/token_store.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * sparse/token_store.c
 *
 * Copyright (C) 2012 Oracle.
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "lib.h"
#include "parse.h"
#include "allocate.h"

struct line {
        struct position pos;
        struct line *prev;
        struct token *token;
        struct line *next;
};

__ALLOCATOR(struct token, "token store", perm_token);
ALLOCATOR(line, "line of tokens");

static struct token *copy_token(struct token *token)
{
        struct token *new;

        new = __alloc_perm_token(0);
        memcpy(new, token, sizeof(*token));
        new->next = NULL;
        return new;
}

static struct line *cursor;

static void find_line(struct position pos)
{
        if (!cursor)
                return;
        if (pos.line == cursor->pos.line)
                return;
        if (pos.line < cursor->pos.line) {
```



```
                if (!cursor->prev)
                        return;
                cursor = cursor->prev;
                find_line(pos);
                return;
        }
        if (!cursor->next)
                return;
        if (pos.line < cursor->next->pos.line)
                return;
        cursor = cursor->next;
        find_line(pos);
}

static void insert_into_line(struct token **current, struct token *new)
{
        if (!*current) {
                *current = new;
                return;
        }

        if (new->pos.pos < (*current)->pos.pos) {
                new->next = *current;
                *current = new;
                return;
        }

        if (new->pos.pos == (*current)->pos.pos)
                return;

        insert_into_line(&(*current)->next, new);
}

static void store_token(struct token *token)
{
        token = copy_token(token);

        find_line(token->pos);

        if (!cursor) {
                cursor = __alloc_line(0);
                cursor->pos = token->pos;
                cursor->token = token;
                return;
        }

        if (token->pos.line < cursor->pos.line) {
                cursor->prev = __alloc_line(0);
                cursor->prev->next = cursor;
                cursor = cursor->prev;
                cursor->pos = token->pos;
                cursor->token = token;
                return;
        }

        if (token->pos.line == cursor->pos.line) {
                insert_into_line(&cursor->token, token);
                return;
        }

        cursor->next = __alloc_line(0);
        cursor->next->prev = cursor;
        cursor = cursor->next;
        cursor->pos = token->pos;
        cursor->token = token;
}
```

```
128 void store_all_tokens(struct token *token)
129 {
130         while (token_type(token) != TOKEN_STREAMEND) {
131                 store_token(token);
132                 token = token->next;
133         }
134 }

136 struct token *first_token_from_line(struct position pos)
137 {
138         find_line(pos);

140         if (!cursor)
141                 return NULL;

143         if (cursor->pos.stream != pos.stream)
144                 return NULL;
145         if (cursor->pos.line != pos.line)
146                 return NULL;

148         return cursor->token;
149 }

151 struct token *pos_get_token(struct position pos)
152 {
153         struct token *token;

155         token = first_token_from_line(pos);
156         while (token) {
157                 if (pos.pos == token->pos.pos)
158                         return token;
159                 if (pos.pos < token->pos.pos)
160                         return NULL;
161                 token = token->next;
162         }
163         return NULL;
164 }

166 char *pos_ident(struct position pos)
167 {
168         struct token *token;

170         token = pos_get_token(pos);
171         if (!token)
172                 return NULL;
173         if (token_type(token) != TOKEN_IDENT)
174                 return NULL;
175         return token->ident->name;
176 }
```

```
**********************************************************
   23932 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/tokenize.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * This is a really stupid C tokenizer. It doesn't do any include
 * files or anything complex at all. That's the preprocessor.
 *
 * Copyright (C) 2003 Transmeta Corp.
 *               2003 Linus Torvalds
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to deal
 * in the Software without restriction, including without limitation the rights
 * to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 * copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 * THE SOFTWARE.
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <stdint.h>

#include "lib.h"
#include "allocate.h"
#include "token.h"
#include "symbol.h"

#define EOF (-1)

int input_stream_nr = 0;
struct stream *input_streams;
static int input_streams_allocated;
unsigned int tabstop = 8;
int no_lineno = 0;

#define BUFSIZE (8192)

typedef struct {
        int fd, offset, size;
        int pos, line, nr;
        int newline, whitespace;
        struct token **tokenlist;
        struct token *token;
        unsigned char *buffer;
} stream_t;

const char *stream_name(int stream)
{
```



```
        if (stream < 0 || stream > input_stream_nr)
                return "<bad stream>";
        return input_streams[stream].name;
}

static struct position stream_pos(stream_t *stream)
{
        struct position pos;
        pos.type = 0;
        pos.stream = stream->nr;
        pos.newline = stream->newline;
        pos.whitespace = stream->whitespace;
        pos.pos = stream->pos;

        pos.line = stream->line;
        if (no_lineno)
                pos.line = 123456;

        pos.noexpand = 0;
        return pos;
}

const char *show_special(int val)
{
        static char buffer[4];

        buffer[0] = val;
        buffer[1] = 0;
        if (val >= SPECIAL_BASE)
                strcpy(buffer, (char *) combinations[val - SPECIAL_BASE]);
        return buffer;
}

const char *show_ident(const struct ident *ident)
{
        static char buffer[256];
        if (!ident)
                return "<noident>";
        sprintf(buffer, "%.*s", ident->len, ident->name);
        return buffer;
}

static char *charstr(char *ptr, unsigned char c, unsigned char escape, unsigned
{
        if (isprint(c)) {
                if (c == escape || c == '\\')
                        *ptr++ = '\\';
                *ptr++ = c;
                return ptr;
        }
        *ptr++ = '\\';
        switch (c) {
        case '\n':
                *ptr++ = 'n';
                return ptr;
        case '\t':
                *ptr++ = 't';
                return ptr;
        }
        if (!isdigit(next))
                return ptr + sprintf(ptr, "%o", c);

        return ptr + sprintf(ptr, "%03o", c);
}

const char *show_string(const struct string *string)
```

```
127 {
128         static char buffer[4 * MAX_STRING + 3];
129         char *ptr;
130         int i;
131
132         if (!string->length)
133                 return "<bad_string>";
134         ptr = buffer;
135         *ptr++ = '"';
136         for (i = 0; i < string->length-1; i++) {
137                 const char *p = string->data + i;
138                 ptr = charstr(ptr, p[0], '"', p[1]);
139         }
140         *ptr++ = '"';
141         *ptr = '\0';
142         return buffer;
143 }
144
145 static const char *show_char(const char *s, size_t len, char prefix, char delim)
146 {
147         static char buffer[MAX_STRING + 4];
148         char *p = buffer;
149         if (prefix)
150                 *p++ = prefix;
151         *p++ = delim;
152         memcpy(p, s, len);
153         p += len;
154         *p++ = delim;
155         *p++ = '\0';
156         return buffer;
157 }
158
159 static const char *quote_char(const char *s, size_t len, char prefix, char delim
160 {
161         static char buffer[2*MAX_STRING + 6];
162         size_t i;
163         char *p = buffer;
164         if (prefix)
165                 *p++ = prefix;
166         if (delim == '"')
167                 *p++ = '\\';
168         *p++ = delim;
169         for (i = 0; i < len; i++) {
170                 if (s[i] == '"' || s[i] == '\\')
171                         *p++ = '\\';
172                 *p++ = s[i];
173         }
174         if (delim == '"')
175                 *p++ = '\\';
176         *p++ = delim;
177         *p++ = '\0';
178         return buffer;
179 }
180
181 const char *show_token(const struct token *token)
182 {
183         static char buffer[256];
184
185         if (!token)
186                 return "<no token>";
187         switch (token_type(token)) {
188         case TOKEN_ERROR:
189                 return "syntax error";
190
191         case TOKEN_EOF:
192                 return "end-of-input";
```



```
193
194         case TOKEN_IDENT:
195                 return show_ident(token->ident);
196
197         case TOKEN_NUMBER:
198                 return token->number;
199
200         case TOKEN_SPECIAL:
201                 return show_special(token->special);
202
203         case TOKEN_CHAR:
204                 return show_char(token->string->data,
205                         token->string->length - 1, 0, '\'');
206         case TOKEN_CHAR_EMBEDDED_0 ... TOKEN_CHAR_EMBEDDED_3:
207                 return show_char(token->embedded,
208                         token_type(token) - TOKEN_CHAR, 0, '\'');
209         case TOKEN_WIDE_CHAR:
210                 return show_char(token->string->data,
211                         token->string->length - 1, 'L', '\'');
212         case TOKEN_WIDE_CHAR_EMBEDDED_0 ... TOKEN_WIDE_CHAR_EMBEDDED_3:
213                 return show_char(token->embedded,
214                         token_type(token) - TOKEN_WIDE_CHAR, 'L', '\'');
215         case TOKEN_STRING:
216                 return show_char(token->string->data,
217                         token->string->length - 1, 0, '"');
218         case TOKEN_WIDE_STRING:
219                 return show_char(token->string->data,
220                         token->string->length - 1, 'L', '"');
221
222         case TOKEN_STREAMBEGIN:
223                 sprintf(buffer, "<beginning of '%s'>", stream_name(token->pos.st
224                 return buffer;
225
226         case TOKEN_STREAMEND:
227                 sprintf(buffer, "<end of '%s'>", stream_name(token->pos.stream))
228                 return buffer;
229
230         case TOKEN_UNTAINT:
231                 sprintf(buffer, "<untaint>");
232                 return buffer;
233
234         case TOKEN_ARG_COUNT:
235                 sprintf(buffer, "<argcnt>");
236                 return buffer;
237
238         default:
239                 sprintf(buffer, "unhandled token type '%d' ", token_type(token))
240                 return buffer;
241         }
242 }
243
244 const char *quote_token(const struct token *token)
245 {
246         static char buffer[256];
247
248         switch (token_type(token)) {
249         case TOKEN_ERROR:
250                 return "syntax error";
251
252         case TOKEN_IDENT:
253                 return show_ident(token->ident);
254
255         case TOKEN_NUMBER:
256                 return token->number;
257
258         case TOKEN_SPECIAL:
```

```
259                 return show_special(token->special);

261         case TOKEN_CHAR:
262                 return quote_char(token->string->data,
263                         token->string->length - 1, 0, '\'');
264         case TOKEN_CHAR_EMBEDDED_0 ... TOKEN_CHAR_EMBEDDED_3:
265                 return quote_char(token->embedded,
266                         token_type(token) - TOKEN_CHAR, 0, '\'');
267         case TOKEN_WIDE_CHAR:
268                 return quote_char(token->string->data,
269                         token->string->length - 1, 'L', '\'');
270         case TOKEN_WIDE_CHAR_EMBEDDED_0 ... TOKEN_WIDE_CHAR_EMBEDDED_3:
271                 return quote_char(token->embedded,
272                         token_type(token) - TOKEN_WIDE_CHAR, 'L', '\'');
273         case TOKEN_STRING:
274                 return quote_char(token->string->data,
275                         token->string->length - 1, 0, '"');
276         case TOKEN_WIDE_STRING:
277                 return quote_char(token->string->data,
278                         token->string->length - 1, 'L', '"');
279         default:
280                 sprintf(buffer, "unhandled token type '%d' ", token_type(token))
281                 return buffer;
282         }
283 }

285 #define HASHED_INPUT_BITS (6)
286 #define HASHED_INPUT (1 << HASHED_INPUT_BITS)
287 #define HASH_PRIME 0x9e370001UL

289 static int input_stream_hashes[HASHED_INPUT] = { [0 ... HASHED_INPUT-1] = -1 };

291 int *hash_stream(const char *name)
292 {
293         uint32_t hash = 0;
294         unsigned char c;

296         while ((c = *name++) != 0)
297                 hash = (hash + (c << 4) + (c >> 4)) * 11;

299         hash *= HASH_PRIME;
300         hash >>= 32 - HASHED_INPUT_BITS;
301         return input_stream_hashes + hash;
302 }

304 int init_stream(const char *name, int fd, const char **next_path)
305 {
306         int stream = input_stream_nr, *hash;
307         struct stream *current;

309         if (stream >= input_streams_allocated) {
310                 int newalloc = stream * 4 / 3 + 10;
311                 input_streams = realloc(input_streams, newalloc * sizeof(struct
312                 if (!input_streams)
313                         die("Unable to allocate more streams space");
314                 input_streams_allocated = newalloc;
315         }
316         current = input_streams + stream;
317         memset(current, 0, sizeof(*current));
318         current->name = name;
319         current->fd = fd;
320         current->next_path = next_path;
321         current->path = NULL;
322         current->constant = CONSTANT_FILE_MAYBE;
323         input_stream_nr = stream+1;
324         hash = hash_stream(name);
```



```
325         current->next_stream = *hash;
326         *hash = stream;
327         return stream;
328 }

330 static struct token * alloc_token(stream_t *stream)
331 {
332         struct token *token = __alloc_token(0);
333         token->pos = stream_pos(stream);
334         return token;
335 }

337 /*
338  *  Argh...  That was surprisingly messy - handling '\r' complicates the
339  *  things a _lot_.
340  */
341 static int nextchar_slow(stream_t *stream)
342 {
343         int offset = stream->offset;
344         int size = stream->size;
345         int c;
346         int spliced = 0, had_cr, had_backslash;

348 restart:
349         had_cr = had_backslash = 0;

351 repeat:
352         if (offset >= size) {
353                 if (stream->fd < 0)
354                         goto got_eof;
355                 size = read(stream->fd, stream->buffer, BUFSIZE);
356                 if (size <= 0)
357                         goto got_eof;
358                 stream->size = size;
359                 stream->offset = offset = 0;
360         }

362         c = stream->buffer[offset++];
363         if (had_cr)
364                 goto check_lf;

366         if (c == '\r') {
367                 had_cr = 1;
368                 goto repeat;
369         }

371 norm:
372         if (!had_backslash) {
373                 switch (c) {
374                 case '\t':
375                         stream->pos += tabstop - stream->pos % tabstop;
376                         break;
377                 case '\n':
378                         stream->line++;
379                         stream->pos = 0;
380                         stream->newline = 1;
381                         break;
382                 case '\\':
383                         had_backslash = 1;
384                         stream->pos++;
385                         goto repeat;
386                 default:
387                         stream->pos++;
388                 }
389         } else {
390                 if (c == '\n') {
```

```
391                         stream->line++;
392                         stream->pos = 0;
393                         spliced = 1;
394                         goto restart;
395                 }
396                 offset--;
397                 c = '\\';
398         }
399 out:
400         stream->offset = offset;

402         return c;

404 check_lf:
405         if (c != '\n')
406                 offset--;
407         c = '\n';
408         goto norm;

410 got_eof:
411         if (had_backslash) {
412                 c = '\\';
413                 goto out;
414         }
415         if (stream->pos)
416                 warning(stream_pos(stream), "no newline at end of file");
417         else if (spliced)
418                 warning(stream_pos(stream), "backslash-newline at end of file");
419         return EOF;
420 }

422 /*
423  *  We want that as light as possible while covering all normal cases.
424  *  Slow path (including the logics with line-splicing and EOF sanity
425  *  checks) is in nextchar_slow().
426  */
427 static inline int nextchar(stream_t *stream)
428 {
429         int offset = stream->offset;

431         if (offset < stream->size) {
432                 int c = stream->buffer[offset++];
433                 static const char special[256] = {
434                         ['\t'] = 1, ['\r'] = 1, ['\n'] = 1, ['\\'] = 1
435                 };
436                 if (!special[c]) {
437                         stream->offset = offset;
438                         stream->pos++;
439                         return c;
440                 }
441         }
442         return nextchar_slow(stream);
443 }

445 struct token eof_token_entry;

447 static struct token *mark_eof(stream_t *stream)
448 {
449         struct token *end;

451         end = alloc_token(stream);
452         token_type(end) = TOKEN_STREAMEND;
453         end->pos.newline = 1;

455         eof_token_entry.next = &eof_token_entry;
456         eof_token_entry.pos.newline = 1;
```



```
458         end->next =  &eof_token_entry;
459         *stream->tokenlist = end;
460         stream->tokenlist = NULL;
461         return end;
462 }

464 static void add_token(stream_t *stream)
465 {
466         struct token *token = stream->token;

468         stream->token = NULL;
469         token->next = NULL;
470         *stream->tokenlist = token;
471         stream->tokenlist = &token->next;
472 }

474 static void drop_token(stream_t *stream)
475 {
476         stream->newline |= stream->token->pos.newline;
477         stream->whitespace |= stream->token->pos.whitespace;
478         stream->token = NULL;
479 }

481 enum {
482         Letter = 1,
483         Digit = 2,
484         Hex = 4,
485         Exp = 8,
486         Dot = 16,
487         ValidSecond = 32,
488         Quote = 64,
489 };

491 static const long cclass[257] = {
492         ['0' + 1 ... '7' + 1] = Digit | Hex,    /* \<octal> */
493         ['8' + 1 ... '9' + 1] = Digit | Hex,
494         ['A' + 1 ... 'D' + 1] = Letter | Hex,
495         ['E' + 1] = Letter | Hex | Exp, /* E<exp> */
496         ['F' + 1] = Letter | Hex,
497         ['G' + 1 ... 'O' + 1] = Letter,
498         ['P' + 1] = Letter | Exp,       /* P<exp> */
499         ['Q' + 1 ... 'Z' + 1] = Letter,
500         ['a' + 1 ... 'b' + 1] = Letter | Hex, /* \a, \b */
501         ['c' + 1 ... 'd' + 1] = Letter | Hex,
502         ['e' + 1] = Letter | Hex | Exp,/* \e, e<exp> */
503         ['f' + 1] = Letter | Hex,       /* \f */
504         ['g' + 1 ... 'm' + 1] = Letter,
505         ['n' + 1] = Letter,     /* \n */
506         ['o' + 1] = Letter,
507         ['p' + 1] = Letter | Exp,       /* p<exp> */
508         ['q' + 1] = Letter,
509         ['r' + 1] = Letter,     /* \r */
510         ['s' + 1] = Letter,
511         ['t' + 1] = Letter,     /* \t */
512         ['u' + 1] = Letter,
513         ['v' + 1] = Letter,     /* \v */
514         ['w' + 1] = Letter,
515         ['x' + 1] = Letter,     /* \x<hex> */
516         ['y' + 1 ... 'z' + 1] = Letter,
517         ['_' + 1] = Letter,
518         ['.' + 1] = Dot | ValidSecond,
519         ['=' + 1] = ValidSecond,
520         ['+' + 1] = ValidSecond,
521         ['-' + 1] = ValidSecond,
522         ['>' + 1] = ValidSecond,
```

```
523         ['<' + 1] = ValidSecond,
524         ['&' + 1] = ValidSecond,
525         ['|' + 1] = ValidSecond,
526         ['#' + 1] = ValidSecond,
527         ['\'' + 1] = Quote,
528         ['"' + 1] = Quote,
529 };

531 /*
532  * pp-number:
533  *      digit
534  *      . digit
535  *      pp-number digit
536  *      pp-number identifier-nodigit
537  *      pp-number e sign
538  *      pp-number E sign
539  *      pp-number p sign
540  *      pp-number P sign
541  *      pp-number .
542  */
543 static int get_one_number(int c, int next, stream_t *stream)
544 {
545         struct token *token;
546         static char buffer[4095];
547         char *p = buffer, *buf, *buffer_end = buffer + sizeof (buffer);
548         int len;

550         *p++ = c;
551         for (;;) {
552                 long class =  cclass[next + 1];
553                 if (!(class & (Dot | Digit | Letter)))
554                         break;
555                 if (p != buffer_end)
556                         *p++ = next;
557                 next = nextchar(stream);
558                 if (class & Exp) {
559                         if (next == '-' || next == '+') {
560                                 if (p != buffer_end)
561                                         *p++ = next;
562                                 next = nextchar(stream);
563                         }
564                 }
565         }

567         if (p == buffer_end) {
568                 sparse_error(stream_pos(stream), "number token exceeds %td chara
569                       buffer_end - buffer);
570                 // Pretend we saw just "1".
571                 buffer[0] = '1';
572                 p = buffer + 1;
573         }

575         *p++ = 0;
576         len = p - buffer;
577         buf = __alloc_bytes(len);
578         memcpy(buf, buffer, len);

580         token = stream->token;
581         token_type(token) = TOKEN_NUMBER;
582         token->number = buf;
583         add_token(stream);

585         return next;
586 }

588 static int eat_string(int next, stream_t *stream, enum token_type type)
```



```
589 {
590         static char buffer[MAX_STRING];
591         struct string *string;
592         struct token *token = stream->token;
593         int len = 0;
594         int escape;
595         int want_hex = 0;
596         char delim = type < TOKEN_STRING ? '\'' : '"';

598         for (escape = 0; escape || next != delim; next = nextchar(stream)) {
599                 if (len < MAX_STRING)
600                         buffer[len] = next;
601                 len++;
602                 if (next == '\n') {
603                         warning(stream_pos(stream),
604                                 "Newline in string or character constant");
605                         if (delim == '\'') /* assume it's lost ' */
606                                 break;
607                 }
608                 if (next == EOF) {
609                         warning(stream_pos(stream),
610                                 "End of file in middle of string");
611                         return next;
612                 }
613                 if (!escape) {
614                         if (want_hex && !(cclass[next + 1] & Hex))
615                                 warning(stream_pos(stream),
616                                         "\\x used with no following hex digits")
617                         want_hex = 0;
618                         escape = next == '\\';
619                 } else {
620                         escape = 0;
621                         want_hex = next == 'x';
622                 }
623         }
624         if (want_hex)
625                 warning(stream_pos(stream),
626                         "\\x used with no following hex digits");
627         if (len > MAX_STRING) {
628                 warning(stream_pos(stream), "string too long (%d bytes, %d bytes
629                 len = MAX_STRING;
630         }
631         if (delim == '\'' && len <= 4) {
632                 if (len == 0) {
633                         sparse_error(stream_pos(stream),
634                                 "empty character constant");
635                         return nextchar(stream);
636                 }
637                 token_type(token) = type + len;
638                 memset(buffer + len, '\0', 4 - len);
639                 memcpy(token->embedded, buffer, 4);
640         } else {
641                 token_type(token) = type;
642                 string = __alloc_string(len+1);
643                 memcpy(string->data, buffer, len);
644                 string->data[len] = '\0';
645                 string->length = len+1;
646                 token->string = string;
647         }

649         /* Pass it on.. */
650         token = stream->token;
651         add_token(stream);
652         return nextchar(stream);
653 }
```

```
655 static int drop_stream_eoln(stream_t *stream)
656 {
657         drop_token(stream);
658         for (;;) {
659                 switch (nextchar(stream)) {
660                 case EOF:
661                         return EOF;
662                 case '\n':
663                         return nextchar(stream);
664                 }
665         }
666 }

668 static int drop_stream_comment(stream_t *stream)
669 {
670         int newline;
671         int next;
672         drop_token(stream);
673         newline = stream->newline;

675         next = nextchar(stream);
676         for (;;) {
677                 int curr = next;
678                 if (curr == EOF) {
679                         warning(stream_pos(stream), "End of file in the middle o
680                         return curr;
681                 }
682                 next = nextchar(stream);
683                 if (curr == '*' && next == '/')
684                         break;
685         }
686         stream->newline = newline;
687         return nextchar(stream);
688 }

690 unsigned char combinations[][4] = COMBINATION_STRINGS;

692 #define NR_COMBINATIONS (SPECIAL_ARG_SEPARATOR - SPECIAL_BASE)

694 /* hash function for two-character punctuators - all give unique values */
695 #define special_hash(c0, c1) (((c0*8+c1*2)+((c0*8+c1*2)>>5))&31)

697 /*
698  * note that we won't get false positives - special_hash(0,0) is 0 and
699  * entry 0 is filled (by +=), so all the missing ones are OK.
700  */
701 static unsigned char hash_results[32][2] = {
702 #define RES(c0, c1) [special_hash(c0, c1)] = {c0, c1}
703         RES('+', '='), /* 00 */
704         RES('/', '='), /* 01 */
705         RES('^', '='), /* 05 */
706         RES('&', '&'), /* 07 */
707         RES('#', '#'), /* 08 */
708         RES('<', '<'), /* 0a */
709         RES('<', '='), /* 0c */
710         RES('!', '='), /* 0e */
711         RES('%', '='), /* 0f */
712         RES('-', '-'), /* 10 */
713         RES('-', '='), /* 11 */
714         RES('-', '>'), /* 13 */
715         RES('=', '='), /* 15 */
716         RES('&', '='), /* 17 */
717         RES('*', '='), /* 18 */
718         RES('.', '.'), /* 1a */
719         RES('+', '+'), /* 1b */
720         RES('|', '='), /* 1c */
```



```
721         RES('>', '='), /* 1d */
722         RES('|', '|'), /* 1e */
723         RES('>', '>')  /* 1f */
724 #undef RES
725 };
726 static int code[32] = {
727 #define CODE(c0, c1, value) [special_hash(c0, c1)] = value
728         CODE('+', '=', SPECIAL_ADD_ASSIGN), /* 00 */
729         CODE('/', '=', SPECIAL_DIV_ASSIGN), /* 01 */
730         CODE('^', '=', SPECIAL_XOR_ASSIGN), /* 05 */
731         CODE('&', '&', SPECIAL_LOGICAL_AND), /* 07 */
732         CODE('#', '#', SPECIAL_HASHHASH), /* 08 */
733         CODE('<', '<', SPECIAL_LEFTSHIFT), /* 0a */
734         CODE('<', '=', SPECIAL_LTE), /* 0c */
735         CODE('!', '=', SPECIAL_NOTEQUAL), /* 0e */
736         CODE('%', '=', SPECIAL_MOD_ASSIGN), /* 0f */
737         CODE('-', '-', SPECIAL_DECREMENT), /* 10 */
738         CODE('-', '=', SPECIAL_SUB_ASSIGN), /* 11 */
739         CODE('-', '>', SPECIAL_DEREFERENCE), /* 13 */
740         CODE('=', '=', SPECIAL_EQUAL), /* 15 */
741         CODE('&', '=', SPECIAL_AND_ASSIGN), /* 17 */
742         CODE('*', '=', SPECIAL_MUL_ASSIGN), /* 18 */
743         CODE('.', '.', SPECIAL_DOTDOT), /* 1a */
744         CODE('+', '+', SPECIAL_INCREMENT), /* 1b */
745         CODE('|', '=', SPECIAL_OR_ASSIGN), /* 1c */
746         CODE('>', '=', SPECIAL_GTE), /* 1d */
747         CODE('|', '|', SPECIAL_LOGICAL_OR), /* 1e */
748         CODE('>', '>', SPECIAL_RIGHTSHIFT)  /* 1f */
749 #undef CODE
750 };

752 static int get_one_special(int c, stream_t *stream)
753 {
754         struct token *token;
755         int next, value, i;

757         next = nextchar(stream);

759         /*
760          * Check for numbers, strings, character constants, and comments
761          */
762         switch (c) {
763         case '.':
764                 if (next >= '0' && next <= '9')
765                         return get_one_number(c, next, stream);
766                 break;
767         case '"':
768                 return eat_string(next, stream, TOKEN_STRING);
769         case '\'':
770                 return eat_string(next, stream, TOKEN_CHAR);
771         case '/':
772                 if (next == '/')
773                         return drop_stream_eoln(stream);
774                 if (next == '*')
775                         return drop_stream_comment(stream);
776         }

778         /*
779          * Check for combinations
780          */
781         value = c;
782         if (cclass[next + 1] & ValidSecond) {
783                 i = special_hash(c, next);
784                 if (hash_results[i][0] == c && hash_results[i][1] == next) {
785                         value = code[i];
786                         next = nextchar(stream);
```

```
787                         if (value >= SPECIAL_LEFTSHIFT &&
788                             next == "=="[value - SPECIAL_LEFTSHIFT]) {
789                                 value += 3;
790                                 next = nextchar(stream);
791                         }
792                 }
793         }

795         /* Pass it on.. */
796         token = stream->token;
797         token_type(token) = TOKEN_SPECIAL;
798         token->special = value;
799         add_token(stream);
800         return next;
801 }

803 #define IDENT_HASH_BITS (13)
804 #define IDENT_HASH_SIZE (1<<IDENT_HASH_BITS)
805 #define IDENT_HASH_MASK (IDENT_HASH_SIZE-1)

807 #define ident_hash_init(c)              (c)
808 #define ident_hash_add(oldhash,c)       ((oldhash)*11 + (c))
809 #define ident_hash_end(hash)            ((((hash) >> IDENT_HASH_BITS) + (hash))

811 static struct ident *hash_table[IDENT_HASH_SIZE];
812 static int ident_hit, ident_miss, idents;

814 void show_identifier_stats(void)
815 {
816         int i;
817         int distribution[100];

819         fprintf(stderr, "identifiers: %d hits, %d misses\n",
820                 ident_hit, ident_miss);

822         for (i = 0; i < 100; i++)
823                 distribution[i] = 0;

825         for (i = 0; i < IDENT_HASH_SIZE; i++) {
826                 struct ident * ident = hash_table[i];
827                 int count = 0;

829                 while (ident) {
830                         count++;
831                         ident = ident->next;
832                 }
833                 if (count > 99)
834                         count = 99;
835                 distribution[count]++;
836         }

838         for (i = 0; i < 100; i++) {
839                 if (distribution[i])
840                         fprintf(stderr, "%2d: %d buckets\n", i, distribution[i]);
841         }
842 }

844 static struct ident *alloc_ident(const char *name, int len)
845 {
846         struct ident *ident = __alloc_ident(len);
847         ident->symbols = NULL;
848         ident->len = len;
849         ident->tainted = 0;
850         memcpy(ident->name, name, len);
851         return ident;
852 }
```



```
854 static struct ident * insert_hash(struct ident *ident, unsigned long hash)
855 {
856         ident->next = hash_table[hash];
857         hash_table[hash] = ident;
858         ident_miss++;
859         return ident;
860 }

862 static struct ident *create_hashed_ident(const char *name, int len, unsigned lon
863 {
864         struct ident *ident;
865         struct ident **p;

867         p = &hash_table[hash];
868         while ((ident = *p) != NULL) {
869                 if (ident->len == (unsigned char) len) {
870                         if (strncmp(name, ident->name, len) != 0)
871                                 goto next;

873                         ident_hit++;
874                         return ident;
875                 }
876 next:
877                 //misses++;
878                 p = &ident->next;
879         }
880         ident = alloc_ident(name, len);
881         *p = ident;
882         ident->next = NULL;
883         ident_miss++;
884         idents++;
885         return ident;
886 }

888 static unsigned long hash_name(const char *name, int len)
889 {
890         unsigned long hash;
891         const unsigned char *p = (const unsigned char *)name;

893         hash = ident_hash_init(*p++);
894         while (--len) {
895                 unsigned int i = *p++;
896                 hash = ident_hash_add(hash, i);
897         }
898         return ident_hash_end(hash);
899 }

901 struct ident *hash_ident(struct ident *ident)
902 {
903         return insert_hash(ident, hash_name(ident->name, ident->len));
904 }

906 struct ident *built_in_ident(const char *name)
907 {
908         int len = strlen(name);
909         return create_hashed_ident(name, len, hash_name(name, len));
910 }

912 struct token *built_in_token(int stream, struct ident *ident)
913 {
914         struct token *token;

916         token = __alloc_token(0);
917         token->pos.stream = stream;
918         token_type(token) = TOKEN_IDENT;
```

```
919         token->ident = ident;
920         return token;
921 }

923 static int get_one_identifier(int c, stream_t *stream)
924 {
925         struct token *token;
926         struct ident *ident;
927         unsigned long hash;
928         char buf[256];
929         int len = 1;
930         int next;

932         hash = ident_hash_init(c);
933         buf[0] = c;
934         for (;;) {
935                 next = nextchar(stream);
936                 if (!(cclass[next + 1] & (Letter | Digit)))
937                         break;
938                 if (len >= sizeof(buf))
939                         break;
940                 hash = ident_hash_add(hash, next);
941                 buf[len] = next;
942                 len++;
943         };
944         if (cclass[next + 1] & Quote) {
945                 if (len == 1 && buf[0] == 'L') {
946                         if (next == '\'')
947                                 return eat_string(nextchar(stream), stream,
948                                                   TOKEN_WIDE_CHAR);
949                         else
950                                 return eat_string(nextchar(stream), stream,
951                                                   TOKEN_WIDE_STRING);
952                 }
953         }
954         hash = ident_hash_end(hash);
955         ident = create_hashed_ident(buf, len, hash);

957         /* Pass it on.. */
958         token = stream->token;
959         token_type(token) = TOKEN_IDENT;
960         token->ident = ident;
961         add_token(stream);
962         return next;
963 }

965 static int get_one_token(int c, stream_t *stream)
966 {
967         long class = cclass[c + 1];
968         if (class & Digit)
969                 return get_one_number(c, nextchar(stream), stream);
970         if (class & Letter)
971                 return get_one_identifier(c, stream);
972         return get_one_special(c, stream);
973 }

975 static struct token *setup_stream(stream_t *stream, int idx, int fd,
976         unsigned char *buf, unsigned int buf_size)
977 {
978         struct token *begin;

980         stream->nr = idx;
981         stream->line = 1;
982         stream->newline = 1;
983         stream->whitespace = 0;
984         stream->pos = 0;
```



```
986         stream->token = NULL;
987         stream->fd = fd;
988         stream->offset = 0;
989         stream->size = buf_size;
990         stream->buffer = buf;

992         begin = alloc_token(stream);
993         token_type(begin) = TOKEN_STREAMBEGIN;
994         stream->tokenlist = &begin->next;
995         return begin;
996 }

998 static struct token *tokenize_stream(stream_t *stream)
999 {
1000         int c = nextchar(stream);
1001         while (c != EOF) {
1002                 if (!isspace(c)) {
1003                         struct token *token = alloc_token(stream);
1004                         stream->token = token;
1005                         stream->newline = 0;
1006                         stream->whitespace = 0;
1007                         c = get_one_token(c, stream);
1008                         continue;
1009                 }
1010                 stream->whitespace = 1;
1011                 c = nextchar(stream);
1012         }
1013         return mark_eof(stream);
1014 }

1016 struct token * tokenize_buffer(void *buffer, unsigned long size, struct token **
1017 {
1018         stream_t stream;
1019         struct token *begin;

1021         begin = setup_stream(&stream, 0, -1, buffer, size);
1022         *endtoken = tokenize_stream(&stream);
1023         return begin;
1024 }

1026 struct token * tokenize(const char *name, int fd, struct token *endtoken, const
1027 {
1028         struct token *begin, *end;
1029         stream_t stream;
1030         unsigned char buffer[BUFSIZE];
1031         int idx;

1033         idx = init_stream(name, fd, next_path);
1034         if (idx < 0) {
1035                 // info(endtoken->pos, "File %s is const", name);
1036                 return endtoken;
1037         }

1039         begin = setup_stream(&stream, idx, fd, buffer, 0);
1040         end = tokenize_stream(&stream);
1041         if (endtoken)
1042                 end->next = endtoken;
1043         return begin;
1044 }
```

```
**********************************************************
    3706 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/unssa.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * UnSSA - translate the SSA back to normal form.
 *
 * For now it's done by replacing to set of copies:
 * 1) For each phi-node, replace all their phisrc by copies to a common
 *    temporary.
 * 2) Replace all the phi-nodes by copies of the temporaries to the phi-node tar
 *    This is node to preserve the semantic of the phi-node (they should all "ex
 *    simultaneously on entry in the basic block in which they belong).
 *
 * This is similar to the "Sreedhar method I" except that the copies to the
 * temporaries are not placed at the end of the predecessor basic blocks, but
 * at the place where the phi-node operands are defined.
 * This is particulary easy since these copies are essentialy already present
 * as the corresponding OP_PHISOURCE.
 *
 * While very simple this method create a lot more copies that really necessary.
 * We eliminate some of these copies but most probably most of them are still
 * useless.
 * Ideally, "Sreedhar method III" should be used:
 * "Translating Out of Static Single Assignment Form", V. C. Sreedhar, R. D.-C.
 * D. M. Gillies and V. Santhanam.  SAS'99, Vol. 1694 of Lecture Notes in Comput
 * Science, Springer-Verlag, pp. 194-210, 1999.
 * But for this we need precise liveness, on each %phi and not only on OP_PHI's
 * target pseudos.
 *
 * Copyright (C) 2005 Luc Van Oostenryck
 */

#include "lib.h"
#include "linearize.h"
#include "allocate.h"
#include "flow.h"
#include <assert.h>


static inline int nbr_pseudo_users(pseudo_t p)
{
        return ptr_list_size((struct ptr_list *)p->users);
}

static int simplify_phi_node(struct instruction *phi, pseudo_t tmp)
{
        pseudo_t target = phi->target;
        struct pseudo_user *pu;
        pseudo_t src;

        // verify if this phi can be simplified
        FOR_EACH_PTR(phi->phi_list, src) {
                struct instruction *def = src->def;

                if (!def)
                        continue;
                if (def->bb == phi->bb)
                        return 0;
        } END_FOR_EACH_PTR(src);

        // no need to make a copy of this one
        // -> replace the target pseudo by the tmp
        FOR_EACH_PTR(target->users, pu) {
```



```
                use_pseudo(pu->insn, tmp, pu->userp);
        } END_FOR_EACH_PTR(pu);

        phi->bb = NULL;
        return 1;
}

static void replace_phi_node(struct instruction *phi)
{
        pseudo_t tmp;
        pseudo_t p;

        tmp = alloc_pseudo(NULL);
        tmp->type = phi->target->type;
        tmp->ident = phi->target->ident;
        tmp->def = NULL;                // defined by all the phisrc

        // can we avoid to make of copy?
        simplify_phi_node(phi, tmp);

        // rewrite all it's phi_src to copy to a new tmp
        FOR_EACH_PTR(phi->phi_list, p) {
                struct instruction *def = p->def;
                pseudo_t src;

                if (p == VOID)
                        continue;

                assert(def->opcode == OP_PHISOURCE);

                def->opcode = OP_COPY;
                def->target = tmp;

                // can we eliminate the copy?
                src = def->phi_src;
                if (src->type != PSEUDO_REG)
                        continue;
                switch (nbr_pseudo_users(src)) {
                        struct instruction *insn;
                case 1:
                        insn = src->def;
                        if (!insn)
                                break;
                        insn->target = tmp;
                case 0:
                        kill_instruction(def);
                        def->bb = NULL;
                }
        } END_FOR_EACH_PTR(p);

        if (!phi->bb)
                return;

        // rewrite the phi node:
        //      phi     %rt, ...
        // to:
        //      copy    %rt, %tmp
        phi->opcode = OP_COPY;
        use_pseudo(phi, tmp, &phi->src);
}

static void rewrite_phi_bb(struct basic_block *bb)
{
        struct instruction *insn;

        // Replace all the phi-nodes by copies of a temporary
```

```
127         // (which represent the set of all the %phi that feed them).
128         // The target pseudo doesn't change.
129         FOR_EACH_PTR(bb->insns, insn) {
130                 if (!insn->bb)
131                         continue;
132                 if (insn->opcode != OP_PHI)
133                         continue;
134                 replace_phi_node(insn);
135         } END_FOR_EACH_PTR(insn);
136 }

138 int unssa(struct entrypoint *ep)
139 {
140         struct basic_block *bb;

142         FOR_EACH_PTR(ep->bbs, bb) {
143                 rewrite_phi_bb(bb);
144         } END_FOR_EACH_PTR(bb);

146         return 0;
147 }
```

```
**********************************************************
      37 Fri Dec 21 15:00:39 2018
new/usr/src/tools/smatch/src/validation/.gitignore
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 # test-suite
2 *.diff
3 *.got
4 *.expected
```

```
**********************************************************
     302 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/Woverride-init-def.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int array[] = {
        [1] = 3,
        [1] = 1,                /* check-should-warn */
};

/*
 * check-name: Woverride-init-def
 * check-command: sparse $file
 *
 * check-error-start
Woverride-init-def.c:2:10: warning: Initializer entry defined twice
Woverride-init-def.c:3:10:   also defined here
 * check-error-end
 */
```

```
**********************************************************
     205 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/Woverride-init-no.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int array[] = {
        [1] = 3,
        [1] = 1,                /* check-should-warn */
};

/*
 * check-name: Woverride-init-no
 * check-command: sparse -Wno-override-init $file
 *
 * check-error-start
 * check-error-end
 */
```

**********
318 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/Woverride-init-yes.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
static int array[] = {
        [1] = 3,
        [1] = 1,                /* check-should-warn */
};

/*
 * check-name: Woverride-init-yes
 * check-command: sparse -Woverride-init $file
 *
 * check-error-start
Woverride-init-yes.c:2:10: warning: Initializer entry defined twice
Woverride-init-yes.c:3:10:   also defined here
 * check-error-end
 */
```

```
**********************************************************
     235 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/Wunknown-attribute-def.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int foo(void) __attribute__((unknown_attribute));

   3 /*
   4  * check-name: warn-unknown-attribute
   5  *
   6  * check-error-start
   7 Wunknown-attribute-def.c:1:37: warning: attribute 'unknown_attribute': unknown a
   8  * check-error-end
   9  */
```

```
**********************************************************
     203 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/Wunknown-attribute-no.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int foo(void) __attribute__((unknown_attribute));
   2
   3 /*
   4  * check-name: warn-unknown-attribute-no
   5  * check-command: sparse -Wno-unknown-attribute $file
   6  *
   7  * check-error-start
   8  * check-error-end
   9  */
```

```
**********************************************************
     290 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/Wunknown-attribute-yes.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int foo(void) __attribute__((unknown_attribute));
   2
   3 /*
   4  * check-name: warn-unknown-attribute-yes
   5  * check-command: sparse -Wunknown-attribute $file
   6  *
   7  * check-error-start
   8 Wunknown-attribute-yes.c:1:37: warning: attribute 'unknown_attribute': unknown a
   9  * check-error-end
  10  */
```

```
**********************************************************
     325 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/__func__.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static void f(void)
 2 {
 3         char *s1 = __func__;
 4         char arr[2 * (sizeof __func__ == 2) - 1];
 5         char *s2 = __func__ __func__;
 6 }
 7 /*
 8  * check-name: __func__
 9  * check-command: sparse -Wall $file
10  *
11  * check-error-start
12 __func__.c:5:29: error: Expected ; at end of declaration
13 __func__.c:5:29: error: got __func__
14  * check-error-end
15  */
```

```
**********************************************************
     534 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/abstract-array-declarator-static.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  2 extern void f1(int g[static 1]);
  3 extern void f2(int g[static restrict 1]);
  4 extern void f3(int g[restrict static 1]);
  5 extern void f4(int g[static restrict static 1]);        /* duplicate static erro
  6 extern void f5(int g[restrict static static 1]);        /* duplicate static erro

  8 /*
  9  * check-name: abstract array declarator static
 10  * check-error-start
 11 abstract-array-declarator-static.c:5:38: error: duplicate array static declarato
 12 abstract-array-declarator-static.c:6:38: error: duplicate array static declarato
 13  * check-error-end
 14  */
```

```
**********************************************************
     444 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/address_space.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __user __attribute__((address_space(1)))

extern int poke_memory(void *addr);

static int sys_do_stuff(void __user *user_addr)
{
        return poke_memory(user_addr);
}
/*
 * check-name: address_space attribute
 *
 * check-error-start
address_space.c:7:28: warning: incorrect type in argument 1 (different address s
address_space.c:7:28:    expected void *addr
address_space.c:7:28:    got void <asn:1>*user_addr
 * check-error-end
 */
```

```
**********************************************************
     238 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/alias-distinct.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern int g;
   2 extern int h;

   4 static int foo(void)
   5 {
   6         g = 1;
   7         h = 2;
   8         return g == 1;
   9 }

  11 /*
  12  * check-name: alias distinct symbols
  13  * check-command: test-linearize $file
  14  * check-output-ignore
  15  *
  16  * check-output-contains: ret\\..* *\\$1
  17  */
```

*********************************************************
*336 Fri Dec 21 15:00:40 2018*
*new/usr/src/tools/smatch/src/validation/alias-mixed.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
 1 extern int g;


 4 static int foo(int *p)
 5 {
 6         *p = 1;
 7         g = 2;
 8         return *p == 1;
 9 }

11 static int bar(int *p)
12 {
13         g = 1;
14         *p = 2;
15         return g == 1;
16 }

18 static void test(void)
19 {
20         foo(&g);
21         bar(&g);
22 }

24 /*
25  * check-name: alias symbol/pointer
26  * check-command: test-linearize $file
27  * check-output-ignore
28  *
29  * check-output-excludes: ret\\..* *\\$1
30  */
```

```
**********************************************************
     221 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/alias-same.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern int g;


   4 static int foo(void)
   5 {
   6         g = 1;
   7         g = 2;
   8         return g != 1;
   9 }

  11 /*
  12  * check-name: alias same symbols
  13  * check-command: test-linearize $file
  14  * check-output-ignore
  15  *
  16  * check-output-contains: ret\\..* *\\$1
  17  */
```

```
**********************************************************
    1304 Fri Dec 21 15:00:40 2018
new/usr/src/tools/smatch/src/validation/alloc-align.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned long int size_t;

/*
 * The alloc_align attribute is used to tell the compiler that the return
 * value points to memory, where the returned pointer minimum alignment is given
 * by one of the functions parameters. GCC uses this information to improve
 * pointer alignment analysis.
 *
 * The function parameter denoting the allocated alignment is specified by one
 * integer argument, whose number is the argument of the attribute. Argument
 * numbering starts at one.
 *
 * For instance,
 *
 *    void* my_memalign(size_t, size_t) __attribute__((alloc_align(1)))
 *
 * declares that my_memalign returns memory with minimum alignment given by
 * parameter 1.
 */

#define __alloc_align(x)  __attribute__((__alloc_align__(x)))

/*
 * The aligned_alloc function allocates space for an object whose alignment is
 * specified by alignment, whose size is specified by size, and whose value is
 * indeterminate. The value of alignment shall be a valid alignment supported
 * by the implementation and the value of size shall be an integral multiple
 * of alignment.
 *
 * The aligned_alloc function returns either a null pointer or a pointer to the
 * allocated space.
 */
void *aligned_alloc(size_t alignment, size_t size) __alloc_align(1);


/*
 * check-name: attribute __alloc_align__
 */
```

```
**********************************************************
    1679 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/alternate-keywords.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 2 extern float strtof(const char *__restrict__ ptr, char **__restrict__ endptr);
 3 extern double strtod(const char *__restrict ptr, char **__restrict endptr);
 4 /* restrict: -std=c99 or -std=gnu99 or -std=c11 */
 5 extern long double strtold(const char *restrict ptr, char **restrict endptr);

 7 extern int (*funcs[])(void);

 9 /* typeof: no -std or -std=gnu90 or -std=gnu99 or -std=gnu11 */
10 extern typeof (funcs[0]) f0;
11 extern __typeof (funcs[1]) f1;
12 extern __typeof__(funcs[2]) f2;

14 typedef unsigned short uint16_t;
15 typedef unsigned int uint32_t;
16 typedef unsigned long long uint64_t;

18 static __inline__ uint16_t swap16(uint16_t val)
19 {
20         return ((((uint16_t)(val) & (uint16_t)0x00ffU) << 8) |
21                 (((uint16_t)(val) & (uint16_t)0xff00U) >> 8));
22 }

24 static __inline uint32_t swap32(uint32_t val)
25 {
26         return ((((uint32_t)(val) & (uint32_t)0x000000ffUL) << 24) |
27                 (((uint32_t)(val) & (uint32_t)0x0000ff00UL) <<  8) |
28                 (((uint32_t)(val) & (uint32_t)0x00ff0000UL) >>  8) |
29                 (((uint32_t)(val) & (uint32_t)0xff000000UL) >> 24));
30 }

32 /* inline: no -std or -std=gnu90 or -std=c99 or -std=c11 */
33 static inline uint64_t swap64(uint64_t val)
34 {
35         return ((((uint64_t)(val) & (uint64_t)0x00000000000000ffULL) << 56) |
36                 (((uint64_t)(val) & (uint64_t)0x000000000000ff00ULL) << 40) |
37                 (((uint64_t)(val) & (uint64_t)0x0000000000ff0000ULL) << 24) |
38                 (((uint64_t)(val) & (uint64_t)0x00000000ff000000ULL) <<  8) |
39                 (((uint64_t)(val) & (uint64_t)0x000000ff00000000ULL) >>  8) |
40                 (((uint64_t)(val) & (uint64_t)0x0000ff0000000000ULL) >> 24) |
41                 (((uint64_t)(val) & (uint64_t)0x00ff000000000000ULL) >> 40) |
42                 (((uint64_t)(val) & (uint64_t)0xff00000000000000ULL) >> 56));
43 }
44 /*
45  * check-name: alternate keywords
46  */
```

```
**********************************************************
     160 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/anon-union.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct s {
        union {
                int val;
        };
};

static struct s foo = { .val = 5, };
/*
 * check-name: test anonymous union initializer
 */
```

```
**********************************************************
     889 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/asm-empty-clobber.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 2 # define __ASM_FORM(x)  " " #x " "
 3 # define JUMP_LABEL_INITIAL_NOP ".byte 0xe9 \n\t .long 0\n\t"
 4 # define __ASM_SEL(a,b) __ASM_FORM(b)
 5 #define _ASM_PTR        __ASM_SEL(.long, .quad)

 7 # define JUMP_LABEL(key, label)                                 \
 8       do {                                                      \
 9               asm goto("1:"                                     \
10                       JUMP_LABEL_INITIAL_NOP                    \
11                       ".pushsection __jump_table,  \"a\" \n\t"\
12                       _ASM_PTR "1b, %l[" #label "], %c0 \n\t" \
13                       ".popsection \n\t"                        \
14                       : :  "i" (key) :  : label);               \
15       } while (0)

17 int main(int argc, char *argv[])
18 {
19       JUMP_LABEL("1", do_trace );
20       return 1;
21 do_trace:
22       return 0;
23 }

25 /*
26  *  check-name: Asm with goto labels.
27  */
```

```
**********************************************************
     661 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/asm-goto-lables.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static inline int __static_cpu_has(unsigned char bit)
{
       asm goto("1: jmp %l[t_no]\n"
                "2:\n"
                ".section .altinstructions,\"a\"\n"
                "\n"
                "1b\n"
                "0\n"         /* no replacement */
                " .byte %P0\n"         /* feature bit */
                " .byte 2b - 1b\n"     /* source len */
                " .byte 0\n"           /* replacement len */
                " .byte 0xff + 0 - (2b-1b)\n"  /* padding */
                ".previous\n"
                : : "i" (bit) : : t_no, ble);
       return 1;
t_no:
       return 0;
}
/*
 *  check-name: Asm with goto labels.
 */
```

**********
172 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/asm-toplevel.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
1 __asm__("/* nothing */");
2 /*
3  * check-name: asm-toplevel.c
4  * check-command: test-linearize $file
5  * check-output-ignore
6  * check-output-contains: asm *".. nothing .."
7  */
```

```
**********************************************************
     380 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/attr-inline.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static inline __attribute__((__always_inline__)) int gt(int lhs, int rhs)
{
        return lhs > rhs;
}

extern inline __attribute__((__gnu_inline__)) int ge(int lhs, int rhs)
{
        return lhs >= rhs;
}

static __attribute__((__warning__("That's junk!"))) __attribute__((__unused__))
__attribute__((__noinline__))
void junk(void)
{
        __asm__("");
}

/*
 * check-name: inline attributes
 */
```

**********
169 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/attr-no_sanitize_address.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#define __no_sanitize_address __attribute__((no_sanitize_address))

static void __no_sanitize_address bar(void)
{
}

/*
 * check-name: attribute no_sanitize_address
 */
```

**********
122 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/attr-noclone.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#define noclone         __attribute__((__noclone__))

static void noclone bar(void)
{
}

/*
 * check-name: attribute noclone
 */
```

```
**********************************************************
     258 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/attr-optimize.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __noclone       __attribute__((__noclone__, __optimize__("no-tracer")))

struct kvm_vcpu;

static void __noclone vmx_vcpu_run(struct kvm_vcpu *vcpu)
{
        __asm__("");
}

extern void *run;
void *run = vmx_vcpu_run;

/*
 * check-name: optimize attributes
 */
```

**********
178 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/attr-warning.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
1 # define __warndecl(name, msg) \
2   extern void name (void) __attribute__((__warning__ (msg)))

4 __warndecl (__warn_func, "warn message");

6 /*
7  * check-name: attribute warning
8  */
```

```
**********************************************************
     162 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/attr_aligned.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 void *foo(void) __attribute__((__assume_aligned__(4096)));
2 void *foo(void) __attribute__((assume_aligned(4096)));
3 /*
4  * check-name: attribute assume_aligned
5  */
```

**********

*237 Fri Dec 21 15:00:41 2018*
*new/usr/src/tools/smatch/src/validation/attr_in_parameter.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
#define A __attribute__((address_space(1)))
static int (A *p);
static int A *q;
static void (*f)(A int *x, A int *y) = (void *)0;
static void g(int A *x)
{
        f(x, x);
        p = q;
}
/*
 * check-name: attribute after ( in direct-declarator
 */
```

```
**********************************************************
     127 Fri Dec 21 15:00:41 2018
new/usr/src/tools/smatch/src/validation/attr_vector_size.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int u32;
typedef u32 __attribute__((vector_size(16))) sse128_t;

/*
 * check-name: attribute vector_size
 */
```

```
**********************************************************
    1192 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/arithmetic-ops.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int add(int x, int y)
 2 {
 3         return x + y;
 4 }

 6 static unsigned int uadd(unsigned int x, unsigned int y)
 7 {
 8         return x + y;
 9 }

11 static float fadd(float x, float y)
12 {
13         return x + y;
14 }

16 static double dadd(double x, double y)
17 {
18         return x + y;
19 }

21 static int sub(int x, int y)
22 {
23         return x - y;
24 }

26 static unsigned int usub(unsigned int x, unsigned int y)
27 {
28         return x - y;
29 }

31 static float fsub(float x, float y)
32 {
33         return x - y;
34 }

36 static double dsub(double x, double y)
37 {
38         return x - y;
39 }

41 static int mul(int x, int y)
42 {
43         return x * y;
44 }

46 static unsigned int umul(unsigned int x, unsigned int y)
47 {
48         return x * y;
49 }

51 static float fmul(float x, float y)
52 {
53         return x * y;
54 }

56 static double dmul(double x, double y)
57 {
58         return x * y;
59 }
```



```
61 static int div(int x, int y)
62 {
63         return x / y;
64 }

66 static unsigned int udiv(unsigned int x, unsigned int y)
67 {
68         return x / y;
69 }

71 static float fdiv(float x, float y)
72 {
73         return x / y;
74 }

76 static double ddiv(double x, double y)
77 {
78         return x / y;
79 }

81 static int mod(int x, int y)
82 {
83         return x % y;
84 }

86 static unsigned int umod(unsigned int x, unsigned int y)
87 {
88         return x % y;
89 }

91 /*
92  * check-name: Arithmetic operator code generation
93  * check-command: sparsec -c $file -o tmp.o
94  */
```

```
**********************************************************
     113 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/array.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static char array[128];

   3 /*
   4  * check-name: Array code generation
   5  * check-command: sparsec -c $file -o tmp.o
   6  */
```

```
**********************************************************
     828 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/bitwise-ops.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int shl(int x, int y)
 2 {
 3         return x << y;
 4 }

 6 static unsigned int ushl(unsigned int x, unsigned int y)
 7 {
 8         return x << y;
 9 }

11 static int shr(int x, int y)
12 {
13         return x >> y;
14 }

16 static unsigned int ushr(unsigned int x, unsigned int y)
17 {
18         return x >> y;
19 }

21 static int and(int x, int y)
22 {
23         return x & y;
24 }

26 static unsigned int uand(unsigned int x, unsigned int y)
27 {
28         return x & y;
29 }

31 static int or(int x, int y)
32 {
33         return x | y;
34 }

36 static unsigned int uor(unsigned int x, unsigned int y)
37 {
38         return x | y;
39 }

41 static int xor(int x, int y)
42 {
43         return x ^ y;
44 }

46 static unsigned int uxor(unsigned int x, unsigned int y)
47 {
48         return x ^ y;
49 }

51 static int not(int x)
52 {
53         return ~x;
54 }

56 static unsigned int unot(unsigned int x)
57 {
58         return ~x;
59 }
```



```
61 /*
62  * check-name: Bitwise operator code generation
63  * check-command: sparsec -c $file -o tmp.o
64  */
```

```
**********************************************************
     143 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/bool-test.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static _Bool return_false(void)
{
        return 0;
}

/*
 * check-name: Boolean type code generation
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
    1070 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/cast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef _Bool bool;
typedef unsigned char uchar;
typedef unsigned short ushort;
typedef unsigned int uint;
typedef unsigned long ulong;
typedef long long longlong;
typedef unsigned long long ulonglong;

#define DEFINE_CAST(from, to)                   \
        static to from##2##to(from x) {         \
                return x;                       \
        }

#define DEFINE_CASTS(from)                      \
        DEFINE_CAST(from, bool)                 \
        DEFINE_CAST(from, char)                 \
        DEFINE_CAST(from, uchar)                \
        DEFINE_CAST(from, short)                \
        DEFINE_CAST(from, ushort)               \
        DEFINE_CAST(from, int)                  \
        DEFINE_CAST(from, uint)                 \
        DEFINE_CAST(from, long)                 \
        DEFINE_CAST(from, ulong)                \
        DEFINE_CAST(from, longlong)             \
        DEFINE_CAST(from, ulonglong)            \
/*
        DEFINE_CAST(from, float)                \
        DEFINE_CAST(from, double)
*/

DEFINE_CASTS(bool)
DEFINE_CASTS(char)
DEFINE_CASTS(uchar)
DEFINE_CASTS(short)
DEFINE_CASTS(ushort)
DEFINE_CASTS(int)
DEFINE_CASTS(uint)
DEFINE_CASTS(long)
DEFINE_CASTS(ulong)
DEFINE_CASTS(longlong)
DEFINE_CASTS(ulonglong)
/*
DEFINE_CASTS(float)
DEFINE_CASTS(double)
*/

/*
 * check-name: Cast code generation
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
    1022 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/cmp-ops.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int sete(int x, int y)
 2 {
 3         return x == y;
 4 }

 6 static int setne(int x, int y)
 7 {
 8         return x != y;
 9 }

11 static int setl(int x, int y)
12 {
13         return x < y;
14 }

16 static int setg(int x, int y)
17 {
18         return x > y;
19 }

21 static int setle(int x, int y)
22 {
23         return x <= y;
24 }

26 static int setge(int x, int y)
27 {
28         return x >= y;
29 }

31 static int setb(unsigned int x, unsigned int y)
32 {
33         return x < y;
34 }

36 static int seta(unsigned int x, unsigned int y)
37 {
38         return x > y;
39 }

41 static int setbe(unsigned int x, unsigned int y)
42 {
43         return x <= y;
44 }

46 static int setae(unsigned int x, unsigned int y)
47 {
48         return x >= y;
49 }

51 static int setfe(float x, float y)
52 {
53         return x == y;
54 }

56 static int setfne(float x, float y)
57 {
58         return x != y;
59 }
```



```
61 static int setfl(float x, float y)
62 {
63         return x < y;
64 }

66 static int setfg(float x, float y)
67 {
68         return x > y;
69 }

71 static int setfle(float x, float y)
72 {
73         return x <= y;
74 }

76 static int setfge(float x, float y)
77 {
78         return x >= y;
79 }

81 /*
82  * check-name: Comparison operator code generation
83  * check-command: sparsec -c $file -o tmp.o
84  */
```

```
**********************************************************
     172 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/extern.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern unsigned long foo;

   3 static unsigned long bar(void)
   4 {
   5         return foo;
   6 }

   8 /*
   9  * check-name: Extern symbol code generation
  10  * check-command: sparsec -c $file -o tmp.o
  11  */
```

**********
196 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/function-ptr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
typedef int (*fn_t)(int x, int y);

static int run(fn_t fn, int x, int y)
{
        return fn(x, y);
}

/*
 * check-name: Function pointer code generation
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
     190 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/hello.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
        puts("hello, world");

        return 0;
}

/*
 * check-name: 'hello, world' code generation
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
     417 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/int-cond.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static long foo(long a, long b, long c)
{
        return a? b:c;
}

static long foo_bool(_Bool a, long b, long c)
{
        return a? b:c;
}

static long bar(long a, long b, long c)
{
        if (a)
                return b;
        else
                return b + c;
}

static long bar_bool(_Bool a, long b, long c)
{
        if (a)
                return b;
        else
                return b + c;
}

/*
 * check-name: Non-bool condition values in branch/select
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
     210 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/load-type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern struct _IO_FILE *stdin;

   3 static void sub(struct _IO_FILE *in) {}

   5 static void test(void) {
   6         sub(stdin);
   7 }

   9 /*
  10  * check-name: Type of loaded objects
  11  * check-command: sparsec -c $file -o tmp.o
  12  */
```

**********
373 Fri Dec 21 15:00:42 2018
new/usr/src/tools/smatch/src/validation/backend/logical-ops.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 static int and_bool(int x, int y)
 2 {
 3         return x && y;
 4 }

 6 static unsigned int uand_bool(unsigned int x, unsigned int y)
 7 {
 8         return x && y;
 9 }

11 static int or_bool(int x, int y)
12 {
13         return x || y;
14 }

16 static unsigned int uor_bool(unsigned int x, unsigned int y)
17 {
18         return x || y;
19 }

21 /*
22  * check-name: Logical operator code generation
23  * check-command: sparsec -c $file -o tmp.o
24  */
```

```
**********************************************************
     218 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/loop.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1
 2 extern int bar (int);
 3
 4 extern int foo (int);
 5
 6 int foo (int x)
 7 {
 8         int y = 0;
 9
10         while (y < 1000) {
11                 y += bar(x);
12         }
13
14         return y;
15 }
16

18 /*
19  * check-name: Loops
20  * check-command: sparsec -c $file -o tmp.o
21  */
```

```
**********************************************************
     183 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/loop2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern int op(void);

   3 static void test(void)
   4 {
   5         int i;
   6         for (i = 0; ; i++) {
   7                 op();
   8         }
   9 }

  11 /*
  12  * check-name: Loops with unused counter
  13  * check-command: sparsec -c $file -o tmp.o
  14  */
```

```
**********************************************************
     168 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/ptrcast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static char *ptrcast(unsigned long *x)
{
        return (unsigned char *) x;
}

/*
 * check-name: Pointer cast code generation
 * check-command: sparsec -c $file -o tmp.o
 */
```

**********
178 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/store-type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
struct foo;
static struct foo *var;

static void set(struct foo *f)
{
	var = f;
}

/*
 * check-name: Type of stored objects
 * check-command: sparsec -c $file -o tmp.o
 */
```

**********
*350 Fri Dec 21 15:00:43 2018*
*new/usr/src/tools/smatch/src/validation/backend/struct-access.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
 1 struct st {
 2         int i, *d;
 3 };

 5 static int load_i(struct st *st)
 6 {
 7         return st->i;
 8 }

10 static void store_i(struct st *st, int i)
11 {
12         st->i = i;
13 }

15 static int *load_d(struct st *st)
16 {
17         return st->d;
18 }

20 static void store_d(struct st *st, int *d)
21 {
22         st->d = d;
23 }

25 /*
26  * check-name: struct access code generation
27  * check-command: sparsec -c $file -o tmp.o
28  */
```

```
**********************************************************
     384 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/struct.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct ctype {
        int                     type;
};

struct symbol {
        void                    *p;
        const char              *name;
        struct ctype            ctype;
        struct symbol           *next_id;
};

struct unnamed {
        struct { int x, y; };
};

static struct symbol sym;
static struct symbol *sym_p;
static struct symbol *sym_q = &sym;

static struct unnamed un;

/*
 * check-name: Struct code generation
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
     368 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/sum.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <stdlib.h>

static int sum(int n)
{
        int i, result = 0;

        for (i = 1; i <= n; ++i)
                result += i;
        return result;
}

int main(int argc, char **argv)
{
        printf("%d\n", sum(5));
        printf("%d\n", sum(100));
        return 0;
}

/*
 * check-name: sum from 1 to n
 * check-command: sparsei $file
 *
 * check-output-start
15
5050
 * check-output-end
 */
```

```
**********************************************************
     183 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/union.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
union foo {
        unsigned long           x;
        unsigned char           y;
        char                    buf[128];
};

static union foo foo;

/*
 * check-name: Union code generation
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
     168 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/backend/void-return-type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static void foo(void)
   2 {
   3 }

   5 static void *bar(void *p)
   6 {
   7         return p;
   8 }

  10 /*
  11  * check-name: void return type code generation
  12  * check-command: sparsec -c $file -o tmp.o
  13  */
```

```
**********************************************************
     367 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/bad-array-designated-initializer.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int a[] = {
        [0] = 0,                // OK
        [\0] = 1,               // KO
};
/*
 * check-name: Bad array designated initializer
 *
 * check-error-start
bad-array-designated-initializer.c:3:10: error: Expected constant expression
bad-array-designated-initializer.c:3:10: error: Expected } at end of initializer
bad-array-designated-initializer.c:3:10: error: got \
 * check-error-end
 */
```

```
**********************************************************
     224 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/bad-assignment.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int foo(int a)
{
        a |=\1;

        return a;
}
/*
 * check-name: bad assignment
 *
 * check-error-start
bad-assignment.c:3:13: error: Expected ; at end of statement
bad-assignment.c:3:13: error: got \
 * check-error-end
 */
```

```
**********************************************************
     278 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/bad-cast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct st;
   2
   3 static int foo(int a)
   4 {
   5         return (struct/st *) a;
   6 }
   7 /*
   8  * check-name: Bad cast syntax
   9  *
  10  * check-error-start
  11 bad-cast.c:5:23: error: expected declaration
  12 bad-cast.c:5:23: error: Expected ) at end of cast operator
  13 bad-cast.c:5:23: error: got /
  14  * check-error-end
  15  */
```

```
**********************************************************
     287 Fri Dec 21 15:00:43 2018
new/usr/src/tools/smatch/src/validation/bad-ternary-cond.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int foo(int a)
{
        return a ?? 1 : 0;
}
/*
 * check-name: Bad ternary syntax
 * check-description: Once caused Sparse to segfault
 * check-error-start
bad-ternary-cond.c:3:19: error: Expected : in conditional expression
bad-ternary-cond.c:3:19: error: got ?
 * check-error-end
 */
```

```
**********************************************************
     213 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/bad-typeof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int fun(void)
{
        typeof() a;
        int b;

        a = b;
}
/*
 * check-name: Bad typeof syntax segfault
 *
 * check-error-start
bad-typeof.c:3:16: error: expected expression after the '(' token
 * check-error-end
 */
```

```
**********************************************************
     189 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/badtype1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static void foo(enum bar baz);

/*
 * check-name: enum not in scope
 * check-known-to-fail
 *
 * check-error-start
badtype1.c:1:22: warning: bad scope for 'enum bar'
 * check-error-end
 */
```

```
**********************************************************
     656 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/badtype2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 //typedef int undef;
   2 extern undef bar(void);
   3 static undef foo(char *c)
   4 {
   5   char p = *c;
   6   switch (p) {
   7   default:
   8     return bar();
   9   }
  10 }

  12 /*
  13  * check-name: missing type
  14  * check-error-start
  15 badtype2.c:2:8: warning: 'undef' has implicit type
  16 badtype2.c:2:14: error: Expected ; at end of declaration
  17 badtype2.c:2:14: error: got bar
  18 badtype2.c:3:14: error: Expected ; at end of declaration
  19 badtype2.c:3:14: error: got foo
  20 badtype2.c:6:3: error: Trying to use reserved word 'switch' as identifier
  21 badtype2.c:7:3: error: not in switch scope
  22 badtype2.c:10:1: error: Expected ; at the end of type declaration
  23 badtype2.c:10:1: error: got }
  24  * check-error-end
  25  */
```

```
**********************************************************
     830 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/badtype3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int
foo (int (*func) (undef, void *), void *data)
{
  int err = 0;
  while (cur) {
    if ((*func) (cur, data))
      break;
  }
  return err;
}

/*
 * check-name: missing type in argument list
 * check-error-start
badtype3.c:2:18: warning: identifier list not in definition
badtype3.c:2:24: error: Expected ) in function declarator
badtype3.c:2:24: error: got ,
badtype3.c:5:3: error: Trying to use reserved word 'while' as identifier
badtype3.c:7:7: error: break/continue not in iterator scope
badtype3.c:9:3: error: Trying to use reserved word 'return' as identifier
badtype3.c:9:10: error: Expected ; at end of declaration
badtype3.c:9:10: error: got err
badtype3.c:10:1: error: Expected ; at the end of type declaration
badtype3.c:10:1: error: got }
badtype3.c:6:11: error: undefined identifier 'func'
 * check-error-end
 */
```

```
**********************************************************
     261 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/badtype4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 void a(void)
 2 {
 3         switch(x) {
 4         case 1:
 5                 break;
 6         }
 7 }
 8 /*
 9  * check-name: switch(bad_type) {...} segfault
10  *
11  * check-error-start
12 badtype4.c:3:16: error: undefined identifier 'x'
13 badtype4.c:4:14: error: incompatible types for 'case' statement
14  * check-error-end
15  */
```

```
**********************************************************
     479 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/badtype5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __force         __attribute__((force))

int foo(int *addr);
int foo(int *addr)
{
        return *(*((typeof(addr) __force *) addr));
}

/*
 * check-name: badtype5.c
 * check-description:
 *      evaluate_dereference() used to miss a call to
 *      examine_symbol_type(). This, in the present, left
 *      a SYM_TYPEOF as type for the last dereferencing
 *      which produced "error: cannot dereference this type".
 *      The presence of the __force and the typeof is needed
 *      to create the situation.
 */
```

```
**********************************************************
      68 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/binary-constant.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 extern int x;

3 int x = 0b11;

5 /*
6  * check-name: binary constant
7  */
```

**********
*1109 Fri Dec 21 15:00:44 2018*
*new/usr/src/tools/smatch/src/validation/bitfield-size.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
struct bfu {
        unsigned int a:4;
        unsigned int  :2;
        unsigned int b:4;
};
unsigned int get__bfu_a(struct bfu bf) { return bf.a; }
unsigned int get__bfu_b(struct bfu bf) { return bf.b; }
unsigned int get_pbfu_a(struct bfu *bf) { return bf->a; }
unsigned int get_pbfu_b(struct bfu *bf) { return bf->b; }


struct bfs {
        signed int a:4;
        signed int  :2;
        signed int b:4;
};
signed int get__bfs_a(struct bfs bf) { return bf.a; }
signed int get__bfs_b(struct bfs bf) { return bf.b; }
signed int get_pbfs_a(struct bfs *bf) { return bf->a; }
signed int get_pbfs_b(struct bfs *bf) { return bf->b; }


struct bfi {
        int a:4;
        int  :2;
        int b:4;
};
unsigned int get__bfi_a(struct bfi bf) { return bf.a; }
unsigned int get__bfi_b(struct bfi bf) { return bf.b; }
unsigned int get_pbfi_a(struct bfi *bf) { return bf->a; }
unsigned int get_pbfi_b(struct bfi *bf) { return bf->b; }

/*
 * check-name: bitfield size
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-pattern-24-times: cast\\.
 * check-output-pattern-12-times: cast\\.4
 * check-output-pattern-6-times: lsr\\..*\\$6
 */
```

**********
*308 Fri Dec 21 15:00:44 2018*
*new/usr/src/tools/smatch/src/validation/bitfields.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
 1 /*
 2  * Al Viro points out that we don't
 3  * do bitfield -> integer promotions
 4  * for array dereferences
 5  *
 6  * "warning: a.c:16:10: incompatible types for operation"
 7  */
 8 static struct {
 9         int x:4;
10 } y;

12 extern int a[];

14 static int b(void)
15 {
16         return a[y.x];
17 }

19 /*
20  * check-name: bitfield to integer promotion
21  */
```

```
**********************************************************
     972 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/bitwise-cast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int u32;
typedef u32 __attribute__((bitwise)) __be32;

/* Implicit casts of 0, legal */
static __be32 foo(void)
{
        __be32 x = 0;

        return 0;
}

/* Explicit cast of 0, legal */
static __be32 bar(void)
{
        return (__be32)0;
}

/* Implicit casts of nonzero, bad */
static __be32 baz(void)
{
        __be32 x = 0x2a;

        return 99;
}

/* Explicit cast of nonzero, bad */
static __be32 quux(void)
{
        return (__be32)1729;
}

/*
 * check-name: conversions to bitwise types
 * check-command: sparse -Wbitwise $file
 * check-error-start
bitwise-cast.c:21:20: warning: incorrect type in initializer (different base typ
bitwise-cast.c:21:20:    expected restricted __be32 [usertype] x
bitwise-cast.c:21:20:    got int
bitwise-cast.c:23:16: warning: incorrect type in return expression (different ba
bitwise-cast.c:23:16:    expected restricted __be32
bitwise-cast.c:23:16:    got int
bitwise-cast.c:29:17: warning: cast to restricted __be32
 * check-error-end
 */
```

```
**********************************************************
     831 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/bool-array.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static _Bool boolarray_d1[1];
static _Bool boolarray_d8[8];
static _Bool boolarray_i2[2] = {
        0,
        1,
};
static int nd1 = sizeof(boolarray_d1);
static int nd8 = sizeof(boolarray_d8);
static int ni2 = sizeof(boolarray_i2);


static long longarray_u2[] = {
        0,
        1,
};
static int nl2 = sizeof(longarray_u2);

/*
 * Used to get "warning: excessive elements in array initializer"
 * for all elements but the first one.
 * Note: only occurs if nbr of elements is a multiple of 8
 *       (if not, theer was another problem)
 */
static _Bool boolarray_u8[] = {
        0,
        1,
        0,
        1,
        0,
        1,
        0,
        1,
};

/*
 * Used to get "error: cannot size expression" for the sizeof.
 */
static _Bool boolarray_u2[] = {
        0,
        1,
};
static int nu2 = sizeof(boolarray_u2);

/*
 * check-name: sizeof(bool array)
 * check-command: sparse -Wno-sizeof-bool $file
 */
```

```
**********************************************************
     631 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/bool-cast-bad.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned short __attribute__((bitwise)) le16;
struct s {
        int a:2;
        int b:2;
        int c:2;
};

static _Bool fresi(le16 a)     { return a; }
static _Bool frese(le16 a)     { return (_Bool)a; }
static _Bool fstsi(struct s a) { return a; }
static _Bool fstse(struct s a) { return (_Bool)a; }

/*
 * check-name: bool-cast-bad.c
 * check-command: sparse $file
 *
 * check-error-start
bool-cast-bad.c:10:41: warning: incorrect type in return expression (different b
bool-cast-bad.c:10:41:    expected bool
bool-cast-bad.c:10:41:    got struct s a
bool-cast-bad.c:11:42: warning: cast from non-scalar
 * check-error-end
 */
```

```
**********************************************************
     635 Fri Dec 21 15:00:44 2018
new/usr/src/tools/smatch/src/validation/bool-cast-explicit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int    u32;
typedef          int    s32;
typedef void *vdp;
typedef int  *sip;
typedef double dbl;
typedef unsigned short __attribute__((bitwise)) le16;

static _Bool fs32(s32 a) { return (_Bool)a; }
static _Bool fu32(u32 a) { return (_Bool)a; }
static _Bool fvdp(vdp a) { return (_Bool)a; }
static _Bool fsip(sip a) { return (_Bool)a; }
static _Bool fdbl(dbl a) { return (_Bool)a; }
static _Bool ffun(void)  { return (_Bool)ffun; }

static _Bool fres(le16 a) { return (_Bool)a; }

/*
 * check-name: bool-cast-explicit
 * check-command: test-linearize -m64 $file
 * check-output-ignore
 * check-output-excludes: cast\\.
 */
```

```
**********************************************************
     629 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/bool-cast-implicit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int    u32;
typedef          int    s32;
typedef void *vdp;
typedef int  *sip;
typedef double dbl;
typedef unsigned short __attribute__((bitwise)) le16;

static _Bool fs32(s32 a) { return a; }
static _Bool fu32(u32 a) { return a; }
static _Bool fvdp(vdp a) { return a; }
static _Bool fsip(sip a) { return a; }
static _Bool fdbl(dbl a) { return a; }
static _Bool ffun(void)  { return ffun; }

static _Bool fres(le16 a) { return a; }

/*
 * check-name: bool-cast-implicit
 * check-command: test-linearize -m64 $file
 * check-output-ignore
 * check-output-excludes: cast\\.
 *
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
    1664 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/bool-cast-restricted.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned   int __attribute__((bitwise)) large_t;
#define LBIT    ((__attribute__((force)) large_t) 1)

_Bool lfoo(large_t x) { return x; }
_Bool qfoo(large_t x) { _Bool r = x; return r; }
_Bool xfoo(large_t x) { return (_Bool)x; }
_Bool lbar(large_t x) { return ~x; }
_Bool qbar(large_t x) { _Bool r = ~x; return r; }
_Bool xbar(large_t x) { return (_Bool)~x; }
_Bool lbaz(large_t x) { return !x; }
_Bool qbaz(large_t x) { _Bool r = !x; return r; }
_Bool xbaz(large_t x) { return (_Bool)!x; }
_Bool lqux(large_t x) { return x & LBIT; }
_Bool qqux(large_t x) { _Bool r = x & LBIT; return r; }
_Bool xqux(large_t x) { return (_Bool)(x & LBIT); }


typedef unsigned short __attribute__((bitwise)) small_t;
#define SBIT    ((__attribute__((force)) small_t) 1)

_Bool sfoo(small_t x) { return x; }
_Bool tfoo(small_t x) { _Bool r = x; return r; }
_Bool zfoo(small_t x) { return (_Bool)x; }
_Bool sbar(small_t x) { return ~x; }
_Bool tbar(small_t x) { _Bool r = ~x; return r; }
_Bool zbar(small_t x) { return (_Bool)~x; }
_Bool sbaz(small_t x) { return !x; }
_Bool tbaz(small_t x) { _Bool r = !x; return r; }
_Bool zbaz(small_t x) { return (_Bool)!x; }
_Bool squx(small_t x) { return x & SBIT; }
_Bool tqux(small_t x) { _Bool r = x & SBIT; return r; }
_Bool zqux(small_t x) { return (_Bool)(x & SBIT); }

/*
 * check-name: bool-cast-restricted.c
 * check-command: sparse -Wno-decl $file
 *
 * check-error-start
bool-cast-restricted.c:24:32: warning: restricted small_t degrades to integer
bool-cast-restricted.c:25:35: warning: restricted small_t degrades to integer
bool-cast-restricted.c:26:33: warning: restricted small_t degrades to integer
 * check-error-end
 */
```

**********
*591 Fri Dec 21 15:00:45 2018*
*new/usr/src/tools/smatch/src/validation/bswap-constant-folding.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
 1 typedef unsigned short __be16;
 2 typedef unsigned short __u16;
 3 typedef unsigned short u16;
 4 #define __force

 6 #define __swab16(x) (__u16)__builtin_bswap16((__u16)(x))
 7 /* the test behaves as though it's always on a little-endian machine */
 8 #define __cpu_to_be16(x) ((__force __be16)__swab16((x)))
 9 #define ___htons(x) __cpu_to_be16(x)
10 #define htons(x) ___htons(x)

12 #define ETH_P_IPV6 0x86DD

14 static u16 protocol;

16 static void test(void)
17 {
18         switch (protocol) {
19         case htons(ETH_P_IPV6):
20                 break;
21         }
22 }

24 /*
25  * check-name: constant folding in bswap builtins
26  * check-error-start
27  * check-error-end
28  */
```

*********************************************************
*344 Fri Dec 21 15:00:45 2018*
*new/usr/src/tools/smatch/src/validation/bug_inline_switch.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
 2 #define __u16 unsigned short
 3 int foo(__u16 n);
 4 static inline __u16 f(__u16 val)
 5 {
 6       return val;
 7 }

 9 static inline unsigned int bar(__u16 n)
10 {
11      switch (n) {
12      case (1 ? 1 : f(1)):
13              return 4;
14      }
15 }

17 int foo(__u16 n)
18 {
19       bar(n);
20       bar(n);
21       return 0;
22 }
23 /*
24  * check-name: inlining switch statement
25  */
```

```
**********************************************************
     104 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/build_smatch_db.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #!/bin/bash

   3 ../smatch --info $* > warns.txt
   4 ../smatch_data/db/create_db.sh warns.txt > /dev/null 2>&1
```

```
**********************************************************
    1805 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/builtin-args-checking.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static unsigned int bad_nbr_args_cte(int a)
 2 {
 3         int r = 0;
 4         r |= __builtin_bswap16();
 5         r |= __builtin_bswap16(1, 2);
 6         r |= __builtin_bswap32();
 7         r |= __builtin_bswap32(1, 2);
 8         r |= __builtin_bswap64();
 9         r |= __builtin_bswap64(1, 2);
10         return r;
11 }

13 static unsigned int bad_nbr_args_var(int a, int b)
14 {
15         int r = 0;
16         r |= __builtin_bswap16();
17         r |= __builtin_bswap16(a, b);
18         r |= __builtin_bswap32();
19         r |= __builtin_bswap32(a, b);
20         r |= __builtin_bswap64();
21         r |= __builtin_bswap64(a, b);
22         return r;
23 }

25 /*
26  * check-name: builtin-args-checking
27  * check-command: sparse $file
28  * check-description: Check that the arguments checking is done
29  *                    for expanded builtins with a prototype.
30  *
31  * check-error-start
32 builtin-args-checking.c:4:31: error: not enough arguments for function __builtin
33 builtin-args-checking.c:5:31: error: too many arguments for function __builtin_b
34 builtin-args-checking.c:6:31: error: not enough arguments for function __builtin
35 builtin-args-checking.c:7:31: error: too many arguments for function __builtin_b
36 builtin-args-checking.c:8:31: error: not enough arguments for function __builtin
37 builtin-args-checking.c:9:31: error: too many arguments for function __builtin_b
38 builtin-args-checking.c:16:31: error: not enough arguments for function __builti
39 builtin-args-checking.c:17:31: error: too many arguments for function __builtin_
40 builtin-args-checking.c:18:31: error: not enough arguments for function __builti
41 builtin-args-checking.c:19:31: error: too many arguments for function __builtin_
42 builtin-args-checking.c:20:31: error: not enough arguments for function __builti
43 builtin-args-checking.c:21:31: error: too many arguments for function __builtin_
44  * check-error-end
45  */
```

*********************************************************
*** 733 Fri Dec 21 15:00:45 2018
*** new/usr/src/tools/smatch/src/validation/builtin-bswap-constant.c
*** 10063 basic support for smatch
*** 10153 checkpaths shouldn't check packaging exceptions
*********************************************************

```
unsigned short bswap16(void);
unsigned short bswap16(void)
{
        return __builtin_bswap16(0x1234);
}

unsigned int bswap32(void);
unsigned int bswap32(void)
{
        return __builtin_bswap32(0x12345678);
}

unsigned long long bswap64(void);
unsigned long long bswap64(void)
{
        return __builtin_bswap64(0x123456789abcdef0ULL);
}

unsigned int half_constant(void);
unsigned int half_constant(void)
{
        int v = 0x12345678;
        return __builtin_bswap32(v);
}
/*
 * check-name: builtin-bswap-constant
 * check-command: test-linearize $file
 *
 * check-output-ignore
 * check-output-excludes: __builtin_bswap
 * check-output-contains:ret.16 *.0x3412
 * check-output-contains:ret.32 *.0x78563412
 * check-output-contains:ret.64 *.0xf0debc9a78563412
 */
```

```
**********************************************************
     781 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/builtin-bswap-variable.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned short     u16;
typedef unsigned int       u32;
typedef unsigned long long u64;

static u16 swap16v(u16 a)
{
        return __builtin_bswap16(a);
}

static u32 swap32v(u64 a)
{
        return __builtin_bswap32(a);
}

static u64 swap64v(u32 a)
{
        return __builtin_bswap64(a);
}

/*
 * check-name: builtin-bswap
 * check-command: test-linearize $file
 * check-description: Check that the right builtin function is called, and
 *                    that the args are correctly promoted or truncated.
 *
 * check-output-ignore
 * check-output-contains:call.16 .* __builtin_bswap16
 * check-output-contains:cast.32 .* (64) %arg1
 * check-output-contains:call.32 .* __builtin_bswap32
 * check-output-contains:cast.64 .* (32) %arg1
 * check-output-contains:call.64 .* __builtin_bswap64
 */
```

**********
739 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/builtin_atomic.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 static void fn(void)
 2 {
 3         static int i, *ptr = (void *)0;

 5         i = __sync_fetch_and_add(ptr, 0);
 6         i = __sync_fetch_and_sub(ptr, 0);
 7         i = __sync_fetch_and_or(ptr, 0);
 8         i = __sync_fetch_and_and(ptr, 0);
 9         i = __sync_fetch_and_xor(ptr, 0);
10         i = __sync_fetch_and_nand(ptr, 0);
11         i = __sync_add_and_fetch(ptr, 0);
12         i = __sync_sub_and_fetch(ptr, 0);
13         i = __sync_or_and_fetch(ptr, 0);
14         i = __sync_and_and_fetch(ptr, 0);
15         i = __sync_xor_and_fetch(ptr, 0);
16         i = __sync_nand_and_fetch(ptr, 0);
17         i = __sync_bool_compare_and_swap(ptr, 0, 1);
18         i = __sync_val_compare_and_swap(ptr, 0, 1);
19         __sync_synchronize();
20         i = __sync_lock_test_and_set(ptr, 0);
21         __sync_lock_release(ptr);
22 }

24 /*
25  * check-name: __builtin_atomic
26  * check-error-start
27  * check-error-end
28  */
```

```
**********************************************************
     225 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/builtin_bswap.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static unsigned short x = __builtin_bswap16(0);
static unsigned int y = __builtin_bswap32(0);
static unsigned long long z = __builtin_bswap64(0);

/*
 * check-name: __builtin_bswap
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
     535 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/builtin_inf.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static double d = __builtin_huge_val();
static float f = __builtin_huge_valf();
static long double l = __builtin_huge_vall();
static double di = __builtin_inf();
static float fi = __builtin_inff();
static long double li = __builtin_infl();
static double dn = __builtin_nan("");
static float fn = __builtin_nanf("");
static long double ln = __builtin_nanl("");
static int inf = __builtin_isinf_sign(0.0);
static int fin = __builtin_isfinite(0.0);
static int nan = __builtin_isnan(0.0);

/*
 * check-name: __builtin INFINITY / nan()
 */
```

```
**********************************************************
    1047 Fri Dec 21 15:00:45 2018
new/usr/src/tools/smatch/src/validation/builtin_safe1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define MY_MACRO(a) do { \
  __builtin_warning(!__builtin_safe_p(a), "Macro argument with side effects: " #
    a;  \
  } while (0)

int g(int);
int h(int) __attribute__((pure));
int i(int) __attribute__((const));

static int foo(int x, int y)
{
  /* unsafe: */
  MY_MACRO(x++);
  MY_MACRO(x+=1);
  MY_MACRO(x=x+1);
  MY_MACRO(x%=y);
  MY_MACRO(x=y);
  MY_MACRO(g(x));
  MY_MACRO((y,g(x)));
  /* safe: */
  MY_MACRO(x+1);
  MY_MACRO(h(x));
  MY_MACRO(i(x));
  return x;
}

/*
 * check-name: __builtin_safe
 * check-error-start
builtin_safe1.c:13:3: warning: Macro argument with side effects: x++
builtin_safe1.c:14:3: warning: Macro argument with side effects: x+=1
builtin_safe1.c:15:3: warning: Macro argument with side effects: x=x+1
builtin_safe1.c:16:3: warning: Macro argument with side effects: x%=y
builtin_safe1.c:17:3: warning: Macro argument with side effects: x=y
builtin_safe1.c:18:3: warning: Macro argument with side effects: g(x)
builtin_safe1.c:19:3: warning: Macro argument with side effects: (y,g(x))
 * check-error-end
 */
```

**********
233 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/builtin_unreachable.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 /* example from gcc documents */
 2
 3 void function_that_never_returns (void);
 4
 5 static int g (int c)
 6 {
 7         if (c)
 8                 return 1;
 9         function_that_never_returns ();
10         __builtin_unreachable ();
11 }
12
13 /*
14  * check-name: __builtin_unreachable()
15  */
```

**********
344 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/builtin_va_arg_pack.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 extern void v(int a, ...);
 2
 3 extern inline __attribute__((__always_inline__)) void f(int a, ...)
 4 {
 5         __SIZE_TYPE__ b = __builtin_va_arg_pack_len();
 6 }
 7
 8 extern inline __attribute__((__always_inline__)) void g(int a, ...)
 9 {
10         v(a, __builtin_va_arg_pack());
11 }
12
13 static void h(void)
14 {
15         f(0, 0);
16         g(0, 0);
17 }
18 /*
19  * check-name: __builtin_va_arg_pack()
20  */
```

```
**********************************************************
     981 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/c11-alignas.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static _Alignas(8)      int v;
   2 static _Alignas(long)   int t;
   3 static _Alignas(void *) int p;
   4 static _Alignas(int[4]) int a;
   5 static _Alignas(0)      int z;
   6 static _Alignas(3)      int bnpow2;
   7 static _Alignas(-1)     int bneg;
   8 static _Alignas(-2)     int bnegpow2;
   9 static _Alignas(v)      int bnc;
  10 static _Alignas(+)      int bsyn;

  12 static int check(void)
  13 {
  14         if (_Alignof(v) != 8)
  15                 return -1;
  16         if (_Alignof(t) != _Alignof(long))
  17                 return -1;
  18         if (_Alignof(p) != _Alignof(void *))
  19                 return -1;
  20         if (_Alignof(a) != _Alignof(int))
  21                 return -1;

  23         return 0;
  24 }

  26 /*
  27  * check-name: c11-alignas
  28  * check-command: test-linearize -std=c11 $file
  29  *
  30  * check-error-start
  31 c11-alignas.c:6:25: warning: non-power-of-2 alignment
  32 c11-alignas.c:7:25: warning: non-positive alignment
  33 c11-alignas.c:8:25: warning: non-positive alignment
  34 c11-alignas.c:9:17: error: bad constant expression
  35 c11-alignas.c:10:17: error: Syntax error in unary expression
  36  * check-error-end
  37  *
  38  * check-output-ignore
  39  * check-output-contains: ret\\.32 *\$0
  40  */
```

```
**********************************************************
     199 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/c11-alignof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int foo(void)
   2 {
   3         return _Alignof(short);
   4 }

   6 /*
   7  * check-name: c11-alignof
   8  * check-command: test-linearize -std=c11 $file
   9  *
  10  * check-output-ignore
  11  * check-output-contains: ret\\.32 *\$2
  12  */
```

```
**********************************************************
     195 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/c11-noreturn.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static _Noreturn void foo(void) { while (1) ; }
   2
   3 /*
   4  * check-name: c11-noreturn
   5  * check-command: test-parsing -std=c11 $file
   6  *
   7  * check-output-ignore
   8  * check-output-contains: \[noreturn\]
   9  */
```

```
**********************************************************
     154 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/c11-stdc-version.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
__STDC_VERSION__

/*
 * check-name: c11-stdc-version
 * check-command: sparse -E -std=c11 $file
 *
 * check-output-start

201112L
 * check-output-end
 */
```

```
**********************************************************
     176 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/c11-thread-local.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static _Thread_local int foo;

   3 /*
   4  * check-name: c11-thread-local
   5  * check-command: test-parsing -std=c11 $file
   6  *
   7  * check-output-ignore
   8  * check-output-contains: \[tls\]
   9  */
```

```
**********************************************************
     986 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/c99-for-loop-decl.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int bad_scope(void)
 2 {
 3         int r = 0;

 5         for (int i = 0; i < 10; i++) {
 6                 r = i;
 7         }

 9         return i;                       /* check-should-fail */
10 }

12 static int c99(void)
13 {
14         int r = 0;

16         for (         int i = 0; i < 10; i++)   /* check-should-pass */
17                 r = i;
18         for (    auto int j = 0; j < 10; j++)   /* check-should-pass */
19                 r = j;
20         for (register int k = 0; k < 10; k++)   /* check-should-pass */
21                 r = k;
22         for (  extern int l = 0; l < 10; l++)   /* check-should-fail */
23                 r = l;
24         for (  extern int m;     m < 10; m++)   /* check-should-fail */
25                 r = m;
26         for (  static int n = 0; n < 10; n++)   /* check-should-fail */
27                 r = n;
28         return r;
29 }

31 /*
32  * check-name: C99 for-loop declarations
33  *
34  * check-error-start
35 c99-for-loop-decl.c:22:27: error: non-local var 'l' in for-loop initializer
36 c99-for-loop-decl.c:24:27: error: non-local var 'm' in for-loop initializer
37 c99-for-loop-decl.c:26:27: error: non-local var 'n' in for-loop initializer
38 c99-for-loop-decl.c:9:16: error: undefined identifier 'i'
39  * check-error-end
40  */
```

```
**********************************************************
     327 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/c99-for-loop.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int c99(void);
int c99(void)
{
        int r = -1;

        for (int i = 0; i < 10; i++) {
                r = i;
        }

        return r;
}

/*
 * check-name: C99 for loop variable declaration
 * check-command: test-linearize $file
 *
 * check-output-ignore
 * check-output-contains: phisrc\\.
 * check-output-contains: phi\\.
 * check-output-contains: add\\.
 */
```

```
**********************************************************
     785 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/calling-convention-attributes.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 extern void __attribute__((cdecl)) c1(void);
 2 typedef void (__attribute__((cdecl)) *c2)(void);
 3 typedef c2 c2ptr;

 5 extern void __attribute__((__cdecl__)) c_1(void);
 6 typedef void (__attribute__((__cdecl__)) *c_2)(void);
 7 typedef c_2 c_2ptr;

 9 extern void __attribute__((stdcall)) s1(void);
10 typedef void (__attribute__((stdcall)) *s2)(void);
11 typedef s2 s2ptr;

13 extern void __attribute__((__stdcall__)) s_1(void);
14 typedef void (__attribute__((__stdcall__)) *s_2)(void);
15 typedef s_2 s_2ptr;

17 extern void __attribute__((fastcall)) f1(void);
18 typedef void (__attribute__((fastcall)) *f2)(void);
19 typedef f2 f2ptr;

21 extern void __attribute__((__fastcall__)) f_1(void);
22 typedef void (__attribute__((__fastcall__)) *f_2)(void);
23 typedef f_2 f_2ptr;
24 /*
25  * check-name: Calling convention attributes
26  */
```

```
**********************************************************
     519 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/cast-constant-to-float.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int uint;
typedef unsigned long ulong;

double f1(void) { return -1; }
double f2(void) { return (double)-1; }
double f3(void) { return -1.0; }

/*
 * check-name: cast-constant-to-float
 * check-command: test-linearize -Wno-decl $file
 *
 * check-output-start
f1:
.L0:
        <entry-point>
        set.64      %r1 <- -1.000000
        ret.64      %r1


f2:
.L2:
        <entry-point>
        set.64      %r3 <- -1.000000
        ret.64      %r3


f3:
.L4:
        <entry-point>
        set.64      %r5 <- -1.000000
        ret.64      %r5


 * check-output-end
 */
```

```
**********************************************************
    5832 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/cast-constants.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int uint;
typedef unsigned long ulong;

static int uint_2_int(void) { return (int)123U; }
static int long_2_int(void) { return (int)123L; }
static int ulong_2_int(void) { return (int)123UL; }
static int vptr_2_int(void) { return (int)((void*)123); }
static int iptr_2_int(void) { return (int)((int*)128); }
static int float_2_int(void) { return (int)1.123F; }
static int double_2_int(void) { return (int)1.123L; }
static uint int_2_uint(void) { return (uint)123; }
static uint long_2_uint(void) { return (uint)123L; }
static uint ulong_2_uint(void) { return (uint)123UL; }
static uint vptr_2_uint(void) { return (uint)((void*)123); }
static uint iptr_2_uint(void) { return (uint)((int*)128); }
static uint float_2_uint(void) { return (uint)1.123F; }
static uint double_2_uint(void) { return (uint)1.123L; }
static long int_2_long(void) { return (long)123; }
static long uint_2_long(void) { return (long)123U; }
static long ulong_2_long(void) { return (long)123UL; }
static long vptr_2_long(void) { return (long)((void*)123); }
static long iptr_2_long(void) { return (long)((int*)128); }
static long float_2_long(void) { return (long)1.123F; }
static long double_2_long(void) { return (long)1.123L; }
static ulong int_2_ulong(void) { return (ulong)123; }
static ulong uint_2_ulong(void) { return (ulong)123U; }
static ulong long_2_ulong(void) { return (ulong)123L; }
static ulong vptr_2_ulong(void) { return (ulong)((void*)123); }
static ulong iptr_2_ulong(void) { return (ulong)((int*)128); }
static ulong float_2_ulong(void) { return (ulong)1.123F; }
static ulong double_2_ulong(void) { return (ulong)1.123L; }
static void * int_2_vptr(void) { return (void *)123; }
static void * uint_2_vptr(void) { return (void *)123U; }
static void * long_2_vptr(void) { return (void *)123L; }
static void * ulong_2_vptr(void) { return (void *)123UL; }
static void * iptr_2_vptr(void) { return (void *)((int*)128); }
static int * int_2_iptr(void) { return (int *)123; }
static int * uint_2_iptr(void) { return (int *)123U; }
static int * long_2_iptr(void) { return (int *)123L; }
static int * ulong_2_iptr(void) { return (int *)123UL; }
static int * vptr_2_iptr(void) { return (int *)((void*)123); }
static float int_2_float(void) { return (float)123; }
static float uint_2_float(void) { return (float)123U; }
static float long_2_float(void) { return (float)123L; }
static float ulong_2_float(void) { return (float)123UL; }
static float double_2_float(void) { return (float)1.123L; }
static double int_2_double(void) { return (double)123; }
static double uint_2_double(void) { return (double)123U; }
static double long_2_double(void) { return (double)123L; }
static double ulong_2_double(void) { return (double)123UL; }
static double float_2_double(void) { return (double)1.123F; }

/*
 * check-name: cast-constants.c
 * check-command: test-linearize -m64 $file
 *
 * check-output-start
uint_2_int:
.L0:
        <entry-point>
```



```
        ret.32      $123


long_2_int:
.L2:
        <entry-point>
        ret.32      $123


ulong_2_int:
.L4:
        <entry-point>
        ret.32      $123


vptr_2_int:
.L6:
        <entry-point>
        ret.32      $123


iptr_2_int:
.L8:
        <entry-point>
        ret.32      $128


float_2_int:
.L10:
        <entry-point>
        ret.32      $1


double_2_int:
.L12:
        <entry-point>
        ret.32      $1


int_2_uint:
.L14:
        <entry-point>
        ret.32      $123


long_2_uint:
.L16:
        <entry-point>
        ret.32      $123


ulong_2_uint:
.L18:
        <entry-point>
        ret.32      $123


vptr_2_uint:
.L20:
        <entry-point>
        ret.32      $123


iptr_2_uint:
.L22:
        <entry-point>
```

```
127         ret.32      $128


130 float_2_uint:
131 .L24:
132         <entry-point>
133         ret.32      $1


136 double_2_uint:
137 .L26:
138         <entry-point>
139         ret.32      $1


142 int_2_long:
143 .L28:
144         <entry-point>
145         ret.64      $123


148 uint_2_long:
149 .L30:
150         <entry-point>
151         ret.64      $123


154 ulong_2_long:
155 .L32:
156         <entry-point>
157         ret.64      $123


160 vptr_2_long:
161 .L34:
162         <entry-point>
163         ret.64      $123


166 iptr_2_long:
167 .L36:
168         <entry-point>
169         ret.64      $128


172 float_2_long:
173 .L38:
174         <entry-point>
175         ret.64      $1


178 double_2_long:
179 .L40:
180         <entry-point>
181         ret.64      $1


184 int_2_ulong:
185 .L42:
186         <entry-point>
187         ret.64      $123


190 uint_2_ulong:
191 .L44:
192         <entry-point>
```



```
193         ret.64      $123


196 long_2_ulong:
197 .L46:
198         <entry-point>
199         ret.64      $123


202 vptr_2_ulong:
203 .L48:
204         <entry-point>
205         ret.64      $123


208 iptr_2_ulong:
209 .L50:
210         <entry-point>
211         ret.64      $128


214 float_2_ulong:
215 .L52:
216         <entry-point>
217         ret.64      $1


220 double_2_ulong:
221 .L54:
222         <entry-point>
223         ret.64      $1


226 int_2_vptr:
227 .L56:
228         <entry-point>
229         ret.64      $123


232 uint_2_vptr:
233 .L58:
234         <entry-point>
235         ret.64      $123


238 long_2_vptr:
239 .L60:
240         <entry-point>
241         ret.64      $123


244 ulong_2_vptr:
245 .L62:
246         <entry-point>
247         ret.64      $123


250 iptr_2_vptr:
251 .L64:
252         <entry-point>
253         ret.64      $128


256 int_2_iptr:
257 .L66:
258         <entry-point>
```

```
259         ret.64      $123


262 uint_2_iptr:
263 .L68:
264         <entry-point>
265         ret.64      $123


268 long_2_iptr:
269 .L70:
270         <entry-point>
271         ret.64      $123


274 ulong_2_iptr:
275 .L72:
276         <entry-point>
277         ret.64      $123


280 vptr_2_iptr:
281 .L74:
282         <entry-point>
283         ret.64      $123


286 int_2_float:
287 .L76:
288         <entry-point>
289         set.32      %r39 <- 123.000000
290         ret.32      %r39


293 uint_2_float:
294 .L78:
295         <entry-point>
296         set.32      %r41 <- 123.000000
297         ret.32      %r41


300 long_2_float:
301 .L80:
302         <entry-point>
303         set.32      %r43 <- 123.000000
304         ret.32      %r43


307 ulong_2_float:
308 .L82:
309         <entry-point>
310         set.32      %r45 <- 123.000000
311         ret.32      %r45


314 double_2_float:
315 .L84:
316         <entry-point>
317         set.32      %r47 <- 1.123000
318         ret.32      %r47


321 int_2_double:
322 .L86:
323         <entry-point>
324         set.64      %r49 <- 123.000000
```



```
325         ret.64      %r49


328 uint_2_double:
329 .L88:
330         <entry-point>
331         set.64      %r51 <- 123.000000
332         ret.64      %r51


335 long_2_double:
336 .L90:
337         <entry-point>
338         set.64      %r53 <- 123.000000
339         ret.64      %r53


342 ulong_2_double:
343 .L92:
344         <entry-point>
345         set.64      %r55 <- 123.000000
346         ret.64      %r55


349 float_2_double:
350 .L94:
351         <entry-point>
352         set.64      %r57 <- 1.123000
353         ret.64      %r57


356  * check-output-end
357  */
```

```
**********************************************************
    6717 Fri Dec 21 15:00:46 2018
new/usr/src/tools/smatch/src/validation/cast-kinds.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int uint;
typedef unsigned long ulong;

static int uint_2_int(uint a) { return (int)a; }
static int long_2_int(long a) { return (int)a; }
static int ulong_2_int(ulong a) { return (int)a; }
static int vptr_2_int(void *a) { return (int)a; }
static int iptr_2_int(int *a) { return (int)a; }
static int float_2_int(float a) { return (int)a; }
static int double_2_int(double a) { return (int)a; }
static uint int_2_uint(int a) { return (uint)a; }
static uint long_2_uint(long a) { return (uint)a; }
static uint ulong_2_uint(ulong a) { return (uint)a; }
static uint vptr_2_uint(void *a) { return (uint)a; }
static uint iptr_2_uint(int *a) { return (uint)a; }
static uint float_2_uint(float a) { return (uint)a; }
static uint double_2_uint(double a) { return (uint)a; }
static long int_2_long(int a) { return (long)a; }
static long uint_2_long(uint a) { return (long)a; }
static long ulong_2_long(ulong a) { return (long)a; }
static long vptr_2_long(void *a) { return (long)a; }
static long iptr_2_long(int *a) { return (long)a; }
static long float_2_long(float a) { return (long)a; }
static long double_2_long(double a) { return (long)a; }
static ulong int_2_ulong(int a) { return (ulong)a; }
static ulong uint_2_ulong(uint a) { return (ulong)a; }
static ulong long_2_ulong(long a) { return (ulong)a; }
static ulong vptr_2_ulong(void *a) { return (ulong)a; }
static ulong iptr_2_ulong(int *a) { return (ulong)a; }
static ulong float_2_ulong(float a) { return (ulong)a; }
static ulong double_2_ulong(double a) { return (ulong)a; }
static void * int_2_vptr(int a) { return (void *)a; }
static void * uint_2_vptr(uint a) { return (void *)a; }
static void * long_2_vptr(long a) { return (void *)a; }
static void * ulong_2_vptr(ulong a) { return (void *)a; }
static void * iptr_2_vptr(int *a) { return (void *)a; }
static int * int_2_iptr(int a) { return (int *)a; }
static int * uint_2_iptr(uint a) { return (int *)a; }
static int * long_2_iptr(long a) { return (int *)a; }
static int * ulong_2_iptr(ulong a) { return (int *)a; }
static int * vptr_2_iptr(void *a) { return (int *)a; }
static float int_2_float(int a) { return (float)a; }
static float uint_2_float(uint a) { return (float)a; }
static float long_2_float(long a) { return (float)a; }
static float ulong_2_float(ulong a) { return (float)a; }
static float double_2_float(double a) { return (float)a; }
static double int_2_double(int a) { return (double)a; }
static double uint_2_double(uint a) { return (double)a; }
static double long_2_double(long a) { return (double)a; }
static double ulong_2_double(ulong a) { return (double)a; }
static double float_2_double(float a) { return (double)a; }

/*
 * check-name: cast-kinds
 * check-command: test-linearize -m64 $file
 *
 * check-output-start
uint_2_int:
.L0:
        <entry-point>
```



```
        ret.32      %arg1


long_2_int:
.L2:
        <entry-point>
        scast.32    %r5 <- (64) %arg1
        ret.32      %r5


ulong_2_int:
.L4:
        <entry-point>
        cast.32     %r8 <- (64) %arg1
        ret.32      %r8


vptr_2_int:
.L6:
        <entry-point>
        cast.32     %r11 <- (64) %arg1
        ret.32      %r11


iptr_2_int:
.L8:
        <entry-point>
        cast.32     %r14 <- (64) %arg1
        ret.32      %r14


float_2_int:
.L10:
        <entry-point>
        ret.32      %arg1


double_2_int:
.L12:
        <entry-point>
        cast.32     %r20 <- (64) %arg1
        ret.32      %r20


int_2_uint:
.L14:
        <entry-point>
        ret.32      %arg1


long_2_uint:
.L16:
        <entry-point>
        scast.32    %r26 <- (64) %arg1
        ret.32      %r26


ulong_2_uint:
.L18:
        <entry-point>
        cast.32     %r29 <- (64) %arg1
        ret.32      %r29


vptr_2_uint:
.L20:
```

```
127         <entry-point>
128         cast.32     %r32 <- (64) %arg1
129         ret.32      %r32


132 iptr_2_uint:
133 .L22:
134         <entry-point>
135         cast.32     %r35 <- (64) %arg1
136         ret.32      %r35


139 float_2_uint:
140 .L24:
141         <entry-point>
142         ret.32      %arg1


145 double_2_uint:
146 .L26:
147         <entry-point>
148         cast.32     %r41 <- (64) %arg1
149         ret.32      %r41


152 int_2_long:
153 .L28:
154         <entry-point>
155         scast.64    %r44 <- (32) %arg1
156         ret.64      %r44


159 uint_2_long:
160 .L30:
161         <entry-point>
162         cast.64     %r47 <- (32) %arg1
163         ret.64      %r47


166 ulong_2_long:
167 .L32:
168         <entry-point>
169         ret.64      %arg1


172 vptr_2_long:
173 .L34:
174         <entry-point>
175         cast.64     %r53 <- (64) %arg1
176         ret.64      %r53


179 iptr_2_long:
180 .L36:
181         <entry-point>
182         cast.64     %r56 <- (64) %arg1
183         ret.64      %r56


186 float_2_long:
187 .L38:
188         <entry-point>
189         cast.64     %r59 <- (32) %arg1
190         ret.64      %r59
```



```
193 double_2_long:
194 .L40:
195         <entry-point>
196         ret.64      %arg1


199 int_2_ulong:
200 .L42:
201         <entry-point>
202         scast.64    %r65 <- (32) %arg1
203         ret.64      %r65


206 uint_2_ulong:
207 .L44:
208         <entry-point>
209         cast.64     %r68 <- (32) %arg1
210         ret.64      %r68


213 long_2_ulong:
214 .L46:
215         <entry-point>
216         ret.64      %arg1


219 vptr_2_ulong:
220 .L48:
221         <entry-point>
222         cast.64     %r74 <- (64) %arg1
223         ret.64      %r74


226 iptr_2_ulong:
227 .L50:
228         <entry-point>
229         cast.64     %r77 <- (64) %arg1
230         ret.64      %r77


233 float_2_ulong:
234 .L52:
235         <entry-point>
236         cast.64     %r80 <- (32) %arg1
237         ret.64      %r80


240 double_2_ulong:
241 .L54:
242         <entry-point>
243         ret.64      %arg1


246 int_2_vptr:
247 .L56:
248         <entry-point>
249         scast.64    %r86 <- (32) %arg1
250         ret.64      %r86


253 uint_2_vptr:
254 .L58:
255         <entry-point>
256         cast.64     %r89 <- (32) %arg1
257         ret.64      %r89
```

```
long_2_vptr:
.L60:
        <entry-point>
        scast.64    %r92 <- (64) %arg1
        ret.64      %r92


ulong_2_vptr:
.L62:
        <entry-point>
        cast.64     %r95 <- (64) %arg1
        ret.64      %r95


iptr_2_vptr:
.L64:
        <entry-point>
        cast.64     %r98 <- (64) %arg1
        ret.64      %r98


int_2_iptr:
.L66:
        <entry-point>
        ptrcast.64  %r101 <- (32) %arg1
        ret.64      %r101


uint_2_iptr:
.L68:
        <entry-point>
        ptrcast.64  %r104 <- (32) %arg1
        ret.64      %r104


long_2_iptr:
.L70:
        <entry-point>
        ptrcast.64  %r107 <- (64) %arg1
        ret.64      %r107


ulong_2_iptr:
.L72:
        <entry-point>
        ptrcast.64  %r110 <- (64) %arg1
        ret.64      %r110


vptr_2_iptr:
.L74:
        <entry-point>
        ptrcast.64  %r113 <- (64) %arg1
        ret.64      %r113


int_2_float:
.L76:
        <entry-point>
        fpcast.32   %r116 <- (32) %arg1
        ret.32      %r116


uint_2_float:
.L78:
```



```
        <entry-point>
        fpcast.32   %r119 <- (32) %arg1
        ret.32      %r119


long_2_float:
.L80:
        <entry-point>
        fpcast.32   %r122 <- (64) %arg1
        ret.32      %r122


ulong_2_float:
.L82:
        <entry-point>
        fpcast.32   %r125 <- (64) %arg1
        ret.32      %r125


double_2_float:
.L84:
        <entry-point>
        fpcast.32   %r128 <- (64) %arg1
        ret.32      %r128


int_2_double:
.L86:
        <entry-point>
        fpcast.64   %r131 <- (32) %arg1
        ret.64      %r131


uint_2_double:
.L88:
        <entry-point>
        fpcast.64   %r134 <- (32) %arg1
        ret.64      %r134


long_2_double:
.L90:
        <entry-point>
        fpcast.64   %r137 <- (64) %arg1
        ret.64      %r137


ulong_2_double:
.L92:
        <entry-point>
        fpcast.64   %r140 <- (64) %arg1
        ret.64      %r140


float_2_double:
.L94:
        <entry-point>
        fpcast.64   %r143 <- (32) %arg1
        ret.64      %r143


 * check-output-end
 */
```

**********

*629 Fri Dec 21 15:00:47 2018*
*new/usr/src/tools/smatch/src/validation/check_byte_count-ice.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
extern void *memset (void *s, int c, int n);

static void foo(void *a)
{
        memset(foo, + ', 20);
}
/*
 * check-name: Segfault in check_byte_count after syntax error
 *
 * check-error-start
check_byte_count-ice.c:6:0: warning: Newline in string or character constant
check_byte_count-ice.c:5:23: warning: multi-character character constant
check_byte_count-ice.c:6:1: error: Expected ) in function call
check_byte_count-ice.c:6:1: error: got }
builtin:0:0: error: Expected } at end of function
builtin:0:0: error: got end-of-input
check_byte_count-ice.c:5:15: error: not enough arguments for function memset
 * check-error-end
 */
```

```
**********************************************************
     707 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/choose_expr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int x = __builtin_choose_expr(0,(char *)0,(void)0);
   2 static int y = __builtin_choose_expr(1,(char *)0,(void)0);
   3 static char s[42];
   4 static int z = 1/(sizeof(__builtin_choose_expr(1,s,0)) - 42);

   6 /*
   7  * check-name: choose expr builtin
   8  * check-error-start
   9 choose_expr.c:1:51: warning: incorrect type in initializer (different base types
  10 choose_expr.c:1:51:    expected int static [signed] [toplevel] x
  11 choose_expr.c:1:51:    got void <noident>
  12 choose_expr.c:2:41: warning: incorrect type in initializer (different base types
  13 choose_expr.c:2:41:    expected int static [signed] [toplevel] y
  14 choose_expr.c:2:41:    got char *<noident>
  15 choose_expr.c:4:17: warning: division by zero
  16  * check-error-end
  17  */
```

```
**********************************************************
     269 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/comma.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static char a[sizeof(char *) + 1];
static char b[1/(sizeof(a) - sizeof(0,a))];
static void f(void)
{
        int c[42];
        typeof((void)0,c) d;
        d = c;
}
/*
 * check-name: Comma and array decay
 * check-description: arguments of comma should degenerate
 */
```

**********
*385 Fri Dec 21 15:00:47 2018*
*new/usr/src/tools/smatch/src/validation/compare-null-to-int.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
static unsigned int comparison = (void *)0 == 1;
/*
 * check-name: Compare null pointer constant to int
 * check-description: Sparse used to allow this.
 *
 * check-error-start
compare-null-to-int.c:1:44: error: incompatible types for operation (==)
compare-null-to-int.c:1:44:    left side has type void *
compare-null-to-int.c:1:44:    right side has type int
 * check-error-end
 */
```

*********************************************************
*295 Fri Dec 21 15:00:47 2018*
*new/usr/src/tools/smatch/src/validation/compound-assign-type.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
static unsigned int foo(unsigned int x, long a)
{
        x /= a;
        return x;
}

/*
 * check-name: compound-assign-type
 * check-command: test-linearize -m64 $file
 * check-output-ignore
 *
 * check-output-excludes: divu\\.32
 * check-output-contains: divs\\.64
 * check-output-contains: scast\\.32
 */
```

**********
477 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/cond-address-array.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 int foo(void) {
 2         extern int a[];

 4         if (a)
 5                 return 1;
 6         return 0;
 7 }

 9 int bar(void) {
10         int a[2];

12         if (a)
13                 return 1;
14         return 0;
15 }

17 /*
18  * check-name: cond-address-array.c
19  * check-command: test-linearize -Wno-decl -Waddress $file
20  * check-output-ignore
21  *
22  * check-error-start
23 cond-address-array.c:4:13: warning: the address of an array will always evaluate
24 cond-address-array.c:12:13: warning: the address of an array will always evaluat
25  * check-error-end
26  */
```

```
**********************************************************
     353 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/cond-address-function.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern void func(void);
   2
   3 int global_function(void)
   4 {
   5         if (func)
   6                 return 1;
   7         return 0;
   8 }
   9
  10 /*
  11  * check-name: cond-address-function
  12  * check-command: test-linearize -Wno-decl -Waddress $file
  13  * check-output-ignore
  14  *
  15  * check-error-start
  16 cond-address-function.c:5:13: warning: the address of a function will always eva
  17  * check-error-end
  18  */
```

```
**********************************************************
     315 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/cond-address.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern void f(void);
   2 extern int a[];

   4 int foo(void) { if (f) return 1; return 0; }
   5 int bar(void) { if (a) return 1; return 0; }
   6 int qux(void) { if (f && a) return 1; return 0; }

   8 /*
   9  * check-name: cond-address.c
  10  * check-command: test-linearize -Wno-decl $file
  11  * check-output-ignore
  12  *
  13  * check-excludes: VOID
  14  */
```

**********
*530 Fri Dec 21 15:00:47 2018*
*new/usr/src/tools/smatch/src/validation/cond-err-expand.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
static inline void f(void)
{
        __builtin_constant_p(0);
}

void foo(void)
{
        0 ? 0 : f();
}

void bar(void)
{
        1 ? f() : 0;
}

/*
 * check-name: cond-err-expand.c
 * check-command: test-linearize -Wno-decl $file
 *
 * check-error-start
cond-err-expand.c:8:11: error: incompatible types in conditional expression (dif
cond-err-expand.c:13:11: error: incompatible types in conditional expression (di
 * check-error-end
 *
 * check-output-ignore
 * check-excludes: call.* __builtin_constant_p
 */
```

**********
494 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/cond_expr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 /*
 2  *  Bug in original tree: (real_v ? : x) had been treated as equivalent of
 3  *  (real_v == 0 ? real_v == 0 : x), which gives the wrong type (and no
 4  *  warning from the testcase below).
 5  */
 6 static int x;
 7 static double y;
 8 int a(void)
 9 {
10         return ~(y ? : x);      /* should warn */
11 }
12 /*
13  * check-name: Two-argument conditional expression types
14  *
15  * check-error-start
16 cond_expr.c:10:16: error: incompatible types for operation (~)
17 cond_expr.c:10:16:    argument has type double
18  * check-error-end
19  */
```

```
**********************************************************
     869 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/cond_expr2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern const int *p;
   2 extern volatile void *q;
   3 extern volatile int *r;
   4 static void f(void)
   5 {
   6         q = 1 ? p : q;  // warn: const volatile void * -> const int *
   7         r = 1 ? r : q;  // OK: volatile void * -> volatile int *
   8         r = 1 ? r : p;  // warn: const volatile int * -> volatile int *
   9 }
  10 /*
  11  * check-name: type of conditional expression
  12  * check-description: Used to miss qualifier mixing and mishandle void *
  13  *
  14  * check-error-start
  15 cond_expr2.c:6:11: warning: incorrect type in assignment (different modifiers)
  16 cond_expr2.c:6:11:    expected void volatile *extern [addressable] [toplevel] q
  17 cond_expr2.c:6:11:    got void const volatile *
  18 cond_expr2.c:8:11: warning: incorrect type in assignment (different modifiers)
  19 cond_expr2.c:8:11:    expected int volatile *extern [addressable] [toplevel] [as
  20 cond_expr2.c:8:11:    got int const volatile *
  21  * check-error-end
  22  */
```

**********
*622 Fri Dec 21 15:00:47 2018*
*new/usr/src/tools/smatch/src/validation/cond_expr3.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
static int icmp = 1 / (sizeof(int) - sizeof(1 > 0));
static int fcmp = 1 / (sizeof(int) - sizeof(1.0 == 2.0 - 1.0));
static int lnot = 1 / (sizeof(int) - sizeof(!!1.0));
static int land = 1 / (sizeof(int) - sizeof(2 && 3));
static int lor  = 1 / (sizeof(int) - sizeof('c' || 1.0f));

/*
 * check-name: result type of relational and logical operators
 *
 * check-error-start
cond_expr3.c:1:21: warning: division by zero
cond_expr3.c:2:21: warning: division by zero
cond_expr3.c:3:21: warning: division by zero
cond_expr3.c:4:21: warning: division by zero
cond_expr3.c:5:21: warning: division by zero
 * check-error-end
 */
```

```
**********************************************************
    1960 Fri Dec 21 15:00:47 2018
new/usr/src/tools/smatch/src/validation/conditional-type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 extern void afun(void);
  2 extern void vcond(void);
  3 static int array[3];

  5 struct state {
  6         int nr:2;
  7 };

  9 enum number {
 10         zero,
 11         one,
 12         two,
 13         many,
 14 };

 16 static int bad_if(struct state s)
 17 {
 18         if (vcond()) return 1;
 19         if (s) return 1;
 20         return 0;
 21 }
 22 static void bad_if2(int *a, int *b)
 23 {
 24         if (vcond()) *a = 1;
 25         *b = 0;
 26 }
 27 static int bad_sel(struct state s)
 28 {
 29         return vcond() ? 1 : 0;
 30         return s ? 1 : 0;
 31 }
 32 static int bad_loop_void(void)
 33 {
 34         while (vcond())
 35                 ;
 36         for (;vcond();)
 37                 ;
 38         do
 39                 ;
 40         while (vcond());
 41         return 0;
 42 }


 45 static int good_if_int(int a, _Bool b, long c, unsigned char d)
 46 {
 47         if (a) return 1;
 48         if (b) return 1;
 49         if (c) return 1;
 50         if (d) return 1;
 51         return 0;
 52 }
 53 static int good_if_float(float a, double b)
 54 {
 55         if (a) return 1;
 56         if (b) return 1;
 57         return 0;
 58 }
 59 static int good_if_enum(void)
 60 {
```



```
 61         if (many) return 1;
 62         return 0;
 63 }
 64 static int good_if_bitfield(struct state s, struct state *p)
 65 {
 66         if (s.nr) return 1;
 67         if (p->nr) return 1;
 68         return 0;
 69 }
 70 static int good_if_ptr(void *ptr)
 71 {
 72         if (ptr) return 1;
 73         if (array) return 1;
 74         if (afun) return 1;
 75         return 0;
 76 }

 78 /*
 79  * check-name: conditional-type
 80  *
 81  * check-error-start
 82 conditional-type.c:18:18: error: incorrect type in conditional
 83 conditional-type.c:18:18:    got void
 84 conditional-type.c:19:13: error: incorrect type in conditional
 85 conditional-type.c:19:13:    got struct state s
 86 conditional-type.c:24:18: error: incorrect type in conditional
 87 conditional-type.c:24:18:    got void
 88 conditional-type.c:29:21: error: incorrect type in conditional
 89 conditional-type.c:29:21:    got void
 90 conditional-type.c:30:16: error: incorrect type in conditional
 91 conditional-type.c:30:16:    got struct state s
 92 conditional-type.c:34:21: error: incorrect type in conditional
 93 conditional-type.c:34:21:    got void
 94 conditional-type.c:36:20: error: incorrect type in conditional
 95 conditional-type.c:36:20:    got void
 96 conditional-type.c:40:21: error: incorrect type in conditional
 97 conditional-type.c:40:21:    got void
 98  * check-error-end
 99  */
```

```
**********************************************************
     382 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constant-suffix-32.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define BIGU 0xfffff00000000000U
#define BIGULL 0xfffff00000000000ULL

static unsigned long long a = BIGU;
static unsigned long long b = BIGULL;

/*
 * check-name: constant-suffix
 * check-command: sparse -m32 -Wconstant-suffix $file
 *
 * check-error-start
constant-suffix-32.c:4:31: warning: constant 0xfffff00000000000U is so big it is
 * check-error-end
 */
```

```
**********************************************************
     364 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constant-suffix-64.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define BIGU 0xfffff00000000000U
#define BIGUL 0xfffff00000000000UL

static unsigned long a = BIGU;
static unsigned long b = BIGUL;

/*
 * check-name: constant-suffix
 * check-command: sparse -m64 -Wconstant-suffix $file
 *
 * check-error-start
constant-suffix-64.c:4:26: warning: constant 0xfffff00000000000U is so big it is
 * check-error-end
 */
```

```
**********************************************************
     482 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-addr-of-static-member.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 struct A {
 2         int a;
 3         int b[2];
 4 };

 6 struct B {
 7         int c;
 8         struct A d;
 9 };

11 static struct B a= {1, {1, {1, 1}}};

13 static int *b = &a.d.a; // OK
14 static int *c = &(&a.d)->a;     // OK
15 static int *d = a.d.b;          // OK
16 static int *e = (&a.d)->b;      // OK
17 static int *f = &a.d.b[1];      // OK
18 static int *g = &(&a.d)->b[1];  // OK

20 /*
21  * check-name: address of static object's member constness verification.
22  * check-command: sparse -Wconstexpr-not-const $file
23  *
24  * check-error-start
25  * check-error-end
26  */
```

```
**********************************************************
     864 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-addr-of-static.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int a = 1;
   2 static int b[2] = {1, 1};
   3 static void c(void) {}

   5 static int *d = &a;             // OK
   6 static int *e = d;              // KO
   7 static int *f = b;              // OK

   9 static void (*g)(void) = c;     // OK
  10 static void (*h)(void) = &c;    // OK

  12 static int *i = &*&a;           // OK
  13 static int *j = &*b;            // OK
  14 static int *k = &*d;            // KO


  17 static void l(void) {
  18         int a = 1;
  19         static int *b = &a;     // KO
  20 }

  22 static void m(void) {
  23         static int a = 1;
  24         static int *b = &a;     // OK
  25 }

  27 /*
  28  * check-name: address of static object constness verification.
  29  * check-command: sparse -Wconstexpr-not-const $file
  30  *
  31  * check-error-start
  32 constexpr-addr-of-static.c:6:17: warning: non-constant initializer for static ob
  33 constexpr-addr-of-static.c:14:19: warning: non-constant initializer for static o
  34 constexpr-addr-of-static.c:19:26: warning: non-constant initializer for static o
  35  * check-error-end
  36  */
```

```
**********************************************************
    1273 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-binop.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int a[] = {
        [0 + 0] = 0,                                            // OK
        [0 + 0.] = 0,                                           // KO
        [(void*)0 + 0] = 0,                                     // KO
        [0 + __builtin_choose_expr(0, 0, 0)] = 0,               // OK
        [0 + __builtin_choose_expr(0, 0., 0)] = 0,              // OK
        [0 + __builtin_choose_expr(0, 0, 0.)] = 0,              // KO
        [0 < 0] = 0,                                            // OK
        [0 < 0.] = 0,                                           // KO
        [0 < __builtin_choose_expr(0, 0, 0)] = 0,               // OK
        [0 < __builtin_choose_expr(0, 0., 0)] = 0,              // OK
        [0 < __builtin_choose_expr(0, 0, 0.)] = 0,              // KO
        [0 && 0] = 0,                                           // OK
        [0 && 0.] = 0,                                          // KO
        [0 && __builtin_choose_expr(0, 0, 0)] = 0,              // OK
        [0 && __builtin_choose_expr(0, 0., 0)] = 0,             // OK
        [0 && __builtin_choose_expr(0, 0, 0.)] = 0,             // KO
        [0 + __builtin_types_compatible_p(int, float)] = 0,     // OK
};

/*
 * check-name: Expression constness propagation in binops and alike
 *
 * check-error-start
constexpr-binop.c:3:12: error: bad constant expression
constexpr-binop.c:4:19: error: bad integer constant expression
constexpr-binop.c:7:12: error: bad constant expression
constexpr-binop.c:9:12: error: bad integer constant expression
constexpr-binop.c:12:12: error: bad integer constant expression
constexpr-binop.c:14:12: error: bad integer constant expression
constexpr-binop.c:17:12: error: bad integer constant expression
 * check-error-end
 */
```

```
**********************************************************
     696 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-cast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int a[] = {
        [(int)0] = 0,           // OK
        [(int)(int)0] = 0,      // OK
        [(int)0.] = 0,          // OK
        [(int)(int)0.] = 0,     // OK
        [(int)__builtin_choose_expr(0, 0, 0)] = 0,      // OK
        [(int)__builtin_choose_expr(0, 0, 0.)] = 0,     // OK

        [(int)(float)0] = 0,    // KO
        [(int)(float)0.] = 0,   // KO

        [(int)(void*)0] = 0,    // KO
        [(int)(void*)0.] = 0,   // KO

};
/*
 * check-name: Expression constness propagation in casts
 *
 * check-error-start
constexpr-cast.c:9:11: error: bad integer constant expression
constexpr-cast.c:10:11: error: bad integer constant expression
constexpr-cast.c:12:11: error: bad integer constant expression
constexpr-cast.c:13:11: error: bad integer constant expression
 * check-error-end
 */
```

```
**********************************************************
     536 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-compound-literal.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int *a = &(int){ 1 };    // OK
 2 static int *b = &(int){ *a };   // KO

 4 static void foo(void)
 5 {
 6         int *b = &(int){ 1 };           // OK
 7         int *c = &(int){ *a };          // OK
 8         static int *d = &(int){ 1 };    // KO
 9 }

11 /*
12  * check-name: compound literal address constness verification
13  * check-command: sparse -Wconstexpr-not-const $file
14  *
15  * check-error-start
16 constexpr-compound-literal.c:2:25: warning: non-constant initializer for static
17 constexpr-compound-literal.c:8:27: warning: non-constant initializer for static
18  * check-error-end
19  */
```

```
**********************************************************
    1359 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-conditional.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int a[] = {
 2         [0 ? : 0] = 0,                                        // OK
 3         [1 ? : 0] = 0,                                        // OK
 4         [0 ? 0 : 0] = 0,                                      // OK
 5         [1 ? 0 : 0] = 0,                                      // OK
 6         [0 ? 0 : __builtin_choose_expr(0, 0, 0)] = 0,         // OK
 7         [1 ? __builtin_choose_expr(0, 0, 0) : 0] = 0,         // OK
 8         [0 ? __builtin_choose_expr(0, 0, 0) : 0] = 0,         // OK
 9         [1 ? 1 : __builtin_choose_expr(0, 0, 0)] = 0,         // OK
10         [__builtin_choose_expr(0, 0, 0) ? : 0] = 0,           // OK
11         [__builtin_choose_expr(0, 0, 1) ? : 0] = 0,           // OK
12         [0. ? : 0] = 0,                                  // KO
13         [0 ? 0. : 0] = 0,                                     // KO
14         [1 ? : 0.] = 0,                                  // KO
15         [__builtin_choose_expr(0, 0., 0) ? : 0] = 0,          // OK
16         [__builtin_choose_expr(0, 0, 0.) ? : 0] = 0,          // KO
17         [0 ? __builtin_choose_expr(0, 0., 0) : 0] = 0,        // OK
18         [0 ? __builtin_choose_expr(0, 0, 0.) : 0] = 0,        // KO
19         [1 ? 0 : __builtin_choose_expr(0, 0., 0)] = 0,        // OK
20         [1 ? 0 : __builtin_choose_expr(0, 0, 0.)] = 0,        // KO
21 };

23 /*
24  * check-name: Expression constness propagation in conditional expressions
25  *
26  * check-error-start
27 constexpr-conditional.c:12:13: error: bad constant expression
28 constexpr-conditional.c:13:19: error: bad constant expression
29 constexpr-conditional.c:14:12: error: bad constant expression
30 constexpr-conditional.c:16:42: error: bad constant expression
31 constexpr-conditional.c:18:48: error: bad constant expression
32 constexpr-conditional.c:20:14: error: bad constant expression
33  * check-error-end
34  */
```

```
**********************************************************
    2144 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-init.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int a = 1;                                       // OK
static int b[2] = {1, 1};                               // OK
static void c(void) {}

struct A {
        int a;
        int b[2];
};

struct B {
        int c;
        struct A d;
};

static struct B d= {1, {1, {1, 1}}};                            // OK
static struct B e= {a, {1, {1, 1}}};                            // KO
static struct B f= {1, {a, {1, 1}}};                            // KO
static struct B g= {1, {1, {a, 1}}};                            // KO
static struct B h= {1, {1, {1, a}}};                            // KO
static struct B i= {.c = 1, .d = {.a = 1, .b = {1, 1}}};        // OK
static struct B j= {.c = a, .d = {.a = 1, .b = {1, 1}}};        // KO
static struct B k= {.c = 1, .d = {.a = a, .b = {1, 1}}};        // KO
static struct B l= {.c = 1, .d = {.a = 1, .b = {a, 1}}};        // KO
static struct B m= {.c = 1, .d = {.a = 1, .b = {1, a}}};        // KO

static int n[] = {a, 1};                                // KO
static int o[] = {1, a};                                // KO
static int p[] = {[0] = a, [1] = 1};                    // KO
static int q[] = {[0] = 1, [1] = a};                    // KO

static void r(void) {
        int a = 0;
        int b = a;              // OK
}

static void s(void) {
        int a = 1;
        static int b = a;       // KO
}

/*
 * check-name: static storage object initializer constness verification.
 * check-command: sparse -Wconstexpr-not-const $file
 *
 * check-error-start
constexpr-init.c:16:21: warning: non-constant initializer for static object
constexpr-init.c:17:25: warning: non-constant initializer for static object
constexpr-init.c:18:29: warning: non-constant initializer for static object
constexpr-init.c:19:32: warning: non-constant initializer for static object
constexpr-init.c:21:26: warning: non-constant initializer for static object
constexpr-init.c:22:40: warning: non-constant initializer for static object
constexpr-init.c:23:49: warning: non-constant initializer for static object
constexpr-init.c:24:52: warning: non-constant initializer for static object
constexpr-init.c:26:19: warning: non-constant initializer for static object
constexpr-init.c:27:22: warning: non-constant initializer for static object
constexpr-init.c:28:25: warning: non-constant initializer for static object
constexpr-init.c:29:34: warning: non-constant initializer for static object
constexpr-init.c:38:24: warning: non-constant initializer for static object
 * check-error-end
 */
```

```
**********************************************************
     222 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-labelref.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static void a(void)
 2 {
 3 label1:
 4         ;
 5         static void *b = &&label1;
 6 }

 8 /*
 9  * check-name: label reference constness verification.
10  * check-command: sparse -Wconstexpr-not-const $file
11  *
12  * check-error-start
13  * check-error-end
14  */
```

```
**********************************************************
     436 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-offsetof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct A {
   2         int a[1];
   3         int b;
   4 };

   6 extern int c;

   8 static int o[] = {
   9         [__builtin_offsetof(struct A, b)] = 0,          // OK
  10         [__builtin_offsetof(struct A, a[0])] = 0,       // OK
  11         [__builtin_offsetof(struct A, a[0*0])] = 0,     // OK
  12         [__builtin_offsetof(struct A, a[c])] = 0        // KO
  13 };

  15 /*
  16  * check-name: __builtin_offsetof() constness verification.
  17  *
  18  * check-error-start
  19 constexpr-offsetof.c:12:39: error: bad constant expression
  20  * check-error-end
  21  */
```

```
**********************************************************
     952 Fri Dec 21 15:00:48 2018
new/usr/src/tools/smatch/src/validation/constexpr-pointer-arith.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int a = 1;
   2 static int b[2] = {1, 1};

   4 static int *c = &b[1];                                  // OK
   5 static int *d = (int*)0 + 1;                            // OK
   6 static int *e = &b[1] + 1;                              // OK
   7 static int *f = b + 1;                                  // OK
   8 static int *g = d + 1;                                  // KO
   9 static int *h = &a + 1;                           // OK
  10 static int *i = &b[1] + 1;                              // OK
  11 static int *j = b + 1;                                  // OK
  12 static int *k = d + 1;                                  // KO
  13 static int *l = &*&b[1];                                // OK
  14 static int *m = &*(&a + 1);                             // OK
  15 static int *n = &*(&b[1] + 1);                          // OK
  16 static int *o = &*(b + 1);                              // OK
  17 static int *p = &*(d + 1);                              // KO

  19 /*
  20  * check-name: pointer arithmetic constness verification.
  21  * check-command: sparse -Wconstexpr-not-const $file
  22  *
  23  * check-error-start
  24 constexpr-pointer-arith.c:8:19: warning: non-constant initializer for static obj
  25 constexpr-pointer-arith.c:12:19: warning: non-constant initializer for static ob
  26 constexpr-pointer-arith.c:17:22: warning: non-constant initializer for static ob
  27  * check-error-end
  28  */
```

```
**********************************************************
     344 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/constexpr-pointer-cast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int *a = (int*)0;        // OK
   2 static int b = 0;
   3 static int *c = (int*)b;        // KO


   6 /*
   7  * check-name: integer literal cast to pointer type constness verification.
   8  * check-command: sparse -Wconstexpr-not-const $file
   9  *
  10  * check-error-start
  11 constexpr-pointer-cast.c:3:18: warning: non-constant initializer for static obje
  12  * check-error-end
  13  */
```

```
**********************************************************
     963 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/constexpr-preop.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int a[] = {
  [+0] = 0,                                     // OK
  [+__builtin_choose_expr(0, 0, 0)] = 0,        // OK
  [+0.] = 0,                                    // KO
  [+__builtin_choose_expr(0, 0, 0.)] = 0,       // KO
  [-0] = 0,                                     // OK
  [-__builtin_choose_expr(0, 0, 0)] = 0,        // OK
  [-0.] = 0,                                    // KO
  [-__builtin_choose_expr(0, 0, 0.)] = 0,       // KO
  [~0] = 0,                                     // OK
  [~__builtin_choose_expr(0, 0, 0)] = 0,        // OK
  [!0] = 0,                                     // OK
  [!__builtin_choose_expr(0, 0, 0)] = 0,        // OK
  [!0.] = 0,                                    // KO
  [!__builtin_choose_expr(0, 0, 0.)] = 0,       // KO
};

/*
 * check-name: Expression constness propagation in preops
 *
 * check-error-start
constexpr-preop.c:4:5: error: bad constant expression
constexpr-preop.c:5:33: error: bad constant expression
constexpr-preop.c:8:4: error: bad constant expression
constexpr-preop.c:9:4: error: bad constant expression
constexpr-preop.c:14:4: error: bad integer constant expression
constexpr-preop.c:15:4: error: bad integer constant expression
 * check-error-end
 */
```

**********
475 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/constexpr-pure-builtin.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
// requires constant integer expressions
static int bar[] = {
        [__builtin_bswap16(0x1234)] = 0,                // OK
        [__builtin_bswap32(0x1234)] = 0,                // OK
        [__builtin_bswap64(0x1234)] = 0,                // OK
};

// requires constant integers
static int foo(unsigned long long a)
{
        switch (a) {
        case __builtin_bswap16(1   <<  8):
        case __builtin_bswap32(2L  << 24):
        case __builtin_bswap64(3LL << 56):
                return 0;
        default:
                return 1;
        }
}

/*
 * check-name: constness of pure/const builtins
 */
```

```
**********************************************************
     191 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/constexpr-string.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 static char *a = "foobar";      // OK

3 /*
4  * check-name: string literal constness verification.
5  * check-command: sparse -Wconstexpr-not-const $file
6  *
7  * check-error-start
8  * check-error-end
9  */
```

```
**********************************************************
     186 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/constexpr-types-compatible-p.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int a[] = {[__builtin_types_compatible_p(int, int)] = 0};

/*
 * check-name: __builtin_types_compatible_p() constness verification.
 *
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
    4520 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/context.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __cond_lock(c) ((c) ? ({ __context__(1); 1; }) : 0)

static void a(void) __attribute__((context(0,1)))
{
        __context__(1);
}

static void r(void) __attribute__((context(1,0)))
{
        __context__(-1);
}

extern int _ca(int fail);
#define ca(fail) __cond_lock(_ca(fail))

static void good_paired1(void)
{
        a();
        r();
}

static void good_paired2(void)
{
        a();
        r();
        a();
        r();
}

static void good_paired3(void)
{
        a();
        a();
        r();
        r();
}

static void good_lock1(void) __attribute__((context(0,1)))
{
        a();
}

static void good_lock2(void) __attribute__((context(0,1)))
{
        a();
        r();
        a();
}

static void good_lock3(void) __attribute__((context(0,1)))
{
        a();
        a();
        r();
}

static void good_unlock1(void) __attribute__((context(1,0)))
{
        r();
}
```



```
static void good_unlock2(void) __attribute__((context(1,0)))
{
        a();
        r();
        r();
}

static void warn_lock1(void)
{
        a();
}

static void warn_lock2(void)
{
        a();
        r();
        a();
}

static void warn_lock3(void)
{
        a();
        a();
        r();
}

static void warn_unlock1(void)
{
        r();
}

static void warn_unlock2(void)
{
        a();
        r();
        r();
}

extern int condition, condition2;

static int good_if1(void)
{
        a();
        if(condition) {
                r();
                return -1;
        }
        r();
        return 0;
}

static void good_if2(void)
{
        if(condition) {
                a();
                r();
        }
}

static void good_if3(void)
{
        a();
        if(condition) {
                a();
                r();
```

```
127         }
128         r();
129 }

131 static int warn_if1(void)
132 {
133         a();
134         if(condition)
135                 return -1;
136         r();
137         return 0;
138 }

140 static int warn_if2(void)
141 {
142         a();
143         if(condition) {
144                 r();
145                 return -1;
146         }
147         return 0;
148 }

150 static void good_while1(void)
151 {
152         a();
153         while(condition)
154                 ;
155         r();
156 }

158 static void good_while2(void)
159 {
160         while(condition) {
161                 a();
162                 r();
163         }
164 }

166 static void good_while3(void)
167 {
168         while(condition) {
169                 a();
170                 r();
171                 if(condition2)
172                         break;
173                 a();
174                 r();
175         }
176 }

178 static void good_while4(void)
179 {
180         a();
181         while(1) {
182                 if(condition2) {
183                         r();
184                         break;
185                 }
186         }
187 }

189 static void good_while5(void)
190 {
191         a();
192         while(1) {
```



```
193                 r();
194                 if(condition2)
195                         break;
196                 a();
197         }
198 }

200 static void warn_while1(void)
201 {
202         while(condition) {
203                 a();
204         }
205 }

207 static void warn_while2(void)
208 {
209         while(condition) {
210                 r();
211         }
212 }

214 static void warn_while3(void)
215 {
216         while(condition) {
217                 a();
218                 if(condition2)
219                         break;
220                 r();
221         }
222 }

224 static void good_goto1(void)
225 {
226     a();
227     goto label;
228 label:
229     r();
230 }

232 static void good_goto2(void)
233 {
234     a();
235     goto label;
236     a();
237     r();
238 label:
239     r();
240 }

242 static void good_goto3(void)
243 {
244     a();
245     if(condition)
246         goto label;
247     a();
248     r();
249 label:
250     r();
251 }

253 static void good_goto4(void)
254 {
255     if(condition)
256         goto label;
257     a();
258     r();
```

```
label:
    ;
}

static void good_goto5(void)
{
    a();
    if(condition)
        goto label;
    r();
    return;
label:
    r();
}

static void warn_goto1(void)
{
    a();
    goto label;
    r();
label:
    ;
}

static void warn_goto2(void)
{
    a();
    goto label;
    r();
label:
    a();
    r();
}

static void warn_goto3(void)
{
    a();
    if(condition)
        goto label;
    r();
label:
    r();
}

static void good_cond_lock1(void)
{
    if(ca(condition)) {
        condition2 = 1; /* do stuff */
        r();
    }
}

static void warn_cond_lock1(void)
{
    if(ca(condition))
        condition2 = 1; /* do stuff */
    r();
}
/*
 * check-name: Check -Wcontext
 *
 * check-error-start
context.c:69:13: warning: context imbalance in 'warn_lock1' - wrong count at exi
context.c:74:13: warning: context imbalance in 'warn_lock2' - wrong count at exi
context.c:81:13: warning: context imbalance in 'warn_lock3' - wrong count at exi
context.c:88:13: warning: context imbalance in 'warn_unlock1' - unexpected unloc
```



```
context.c:93:13: warning: context imbalance in 'warn_unlock2' - unexpected unloc
context.c:131:12: warning: context imbalance in 'warn_if1' - wrong count at exit
context.c:140:12: warning: context imbalance in 'warn_if2' - different lock cont
context.c:202:9: warning: context imbalance in 'warn_while1' - different lock co
context.c:210:17: warning: context imbalance in 'warn_while2' - unexpected unloc
context.c:216:9: warning: context imbalance in 'warn_while3' - wrong count at ex
context.c:274:13: warning: context imbalance in 'warn_goto1' - wrong count at ex
context.c:283:13: warning: context imbalance in 'warn_goto2' - wrong count at ex
context.c:300:5: warning: context imbalance in 'warn_goto3' - different lock con
context.c:315:5: warning: context imbalance in 'warn_cond_lock1' - different loc
 * check-error-end
 */
```

**********
247 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/crash-add-doms.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 char a;
 2 int b;
 3 void c(void)
 4 {
 5         if (0) {
 6                 char *d;
 7                 for (;;)
 8                         for (;;)
 9 e:
10                                 *d *= (a && 0) ^ b && *d;
11         }
12         goto e;
13 }


14 /*
15  * check-name: crash add-doms
16  * check-command: test-linearize $file
17  *
18  * check-error-ignore
19  * check-output-ignore
20  */
```

```
**********************************************************
     138 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/crash-bb_target.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
a() {
  &&b

/*
 * check-name: crash bb_target
 * check-command: test-linearize $file
 *
 * check-error-ignore
 * check-output-ignore
 */
```

```
**********************************************************
     168 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/crash-ep-active.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int a(int b)
{
        return 0( && b;
}

/*
 * check-name: crash ep->active
 * check-command: test-linearize $file
 *
 * check-error-ignore
 * check-output-ignore
 */
```

```
**********************************************************
     417 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/crash-ptrlist.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
a;
char b;
c() {
  while () {
    int *d;
    while () {
      char *e = &b;
      if (a ?: (*d = f || *0) || g) {
        if
          ;
      } else {
        int h = 0;
        if (j) {
          char **i = &e;
          if (0 ? 0 : 0 ?: (**i = 1 || 0 && 0))            h ?: (*e = i && &h

/*
 * check-name: crash ptrlist
 * check-command: test-linearize $file
 *
 * check-error-ignore
 * check-output-ignore
 */
```

```
**********************************************************
     290 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/crash-rewrite-branch.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 void a(int c, int e)
 2 {
 3         for(;                   b; c ;

 5         if (()) {
 6                 unsigned short d = e;
 7                 if (())
 8                         while ()
 9                                 ;
10                 &d;
11         }

13         if (()) {
14                 int f = &f;
15         }
16 }

18 /*
19  * check-name: crash rewrite_branch
20  * check-command: test-linearize $file
21  *
22  * check-error-ignore
23  * check-output-ignore
24  */
```

```
**********************************************************
     598 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/crazy02-not-so.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 int foo(int *ptr, int i)
 2 {
 3         int *p;

 5         switch (i - i) {                // will be optimized to 0
 6         case 0:
 7                 return 0;
 8         case 1:                         // will be optimized away
 9                 p = ptr;
10                 do {                    // will be an unreachable loop
11                         *p++ = 123;
12                 } while (--i);
13                 break;
14         }

16         return 1;
17 }

19 int bar(int *ptr, int i)
20 {
21         int *p;

23         switch (i - i) {                // will be optimized to 0
24         case 0:
25                 return 0;
26         case 1:                         // will be optimized away
27                                         // p is uninitialized
28                 do {                    // will be an unreachable loop
29                         *p++ = 123;
30                 } while (--i);
31                 break;
32         }

34         return 1;
35 }

37 /*
38  * check-name: crazy02-not-so.c
39  * check-command: sparse -Wno-decl $file
40  */
```

```
**********************************************************
     357 Fri Dec 21 15:00:49 2018
new/usr/src/tools/smatch/src/validation/crazy03.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 extern char a;
 2 extern int b;
 3 extern char *c, *d;
 4 extern void e(void);
 5 extern void f(char *);

 7 int g(int h);
 8 int g(int h)
 9 {
10         if (h > 1)
11                 e();
12         if (h > 1)
13                 return 0;
14         for (;;) {
15                 if (a) {
16                         while (c) ;
17                         b = 0;
18                 } else {
19                         c = (void*)0;
20                         b = 1;
21                 }
22                 if (b) {
23                         f(c);
24                         continue;
25                 }
26                 d = c;
27                 while (*c++) ;
28         }
29 }

31 /*
32  * check-name: crazy03.c
33  */
```

```
**********************************************************
     257 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/declaration-after-statement-ansi.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static void func (int i)
   2 {
   3         i;
   4         int j = i;
   5 }
   6 /*
   7  * check-name: declaration after statement (ANSI)
   8  * check-command: sparse -ansi $file
   9  * check-error-start
  10 declaration-after-statement-ansi.c:4:9: warning: mixing declarations and code
  11  * check-error-end
  12  */
```

```
**********************************************************
     258 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/declaration-after-statement-c89.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static void func (int i)
 2 {
 3         i;
 4         int j = i;
 5 }
 6 /*
 7  * check-name: declaration after statement (C89)
 8  * check-command: sparse -std=c89 $file
 9  * check-error-start
10 declaration-after-statement-c89.c:4:9: warning: mixing declarations and code
11  * check-error-end
12  */
```

```
**********************************************************
     141 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/declaration-after-statement-c99.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 static void func (int i)
2 {
3         i;
4         int j = i;
5 }
6 /*
7  * check-name: declaration after statement (C99)
8  * check-command: sparse -std=c99 $file
9  */
```

```
**********************************************************
     136 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/declaration-after-statement-default.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 static void func (int i)
2 {
3         i;
4         int j = i;
5 }
6 /*
7  * check-name: declaration after statement (default)
8  * check-command: sparse $file
9  */
```

```
**********************************************************
     153 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/definitions.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static inline int f(void);
 2 static int g(void)
 3 {
 4         return f();
 5 }
 6 static inline int f(void)
 7 {
 8         return 0;
 9 }
10 /*
11  * check-name: finding definitions
12  */
```

```
**********************************************************
    7254 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/designated-init.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct s1 {
        int x;
        int y;
};

struct s2 {
        int x;
        int y;
} __attribute__((designated_init));

struct nest1 {
        struct s1 s1;
        struct s2 s2;
};

struct nest2 {
        struct s1 s1;
        struct s2 s2;
} __attribute__((designated_init));

static struct s1 s1_positional = { 5, 10 };
static struct s1 s1_designated = { .x = 5, .y = 10 };
static struct s2 s2_positional = { 5, 10 };
static struct s2 s2_designated = { .x = 5, .y = 10 };
static struct nest1 nest1_positional = {
        { 5, 10 },
        { 5, 10 },
};
static struct nest1 nest1_designated_outer = {
        .s1 = { 5, 10 },
        .s2 = { 5, 10 },
};
static struct nest1 nest1_designated_inner = {
        { .x = 5, .y = 10 },
        { .x = 5, .y = 10 },
};
static struct nest1 nest1_designated_both = {
        .s1 = { .x = 5, .y = 10 },
        .s2 = { .x = 5, .y = 10 },
};
static struct nest2 nest2_positional = {
        { 5, 10 },
        { 5, 10 },
};
static struct nest2 nest2_designated_outer = {
        .s1 = { 5, 10 },
        .s2 = { 5, 10 },
};
static struct nest2 nest2_designated_inner = {
        { .x = 5, .y = 10 },
        { .x = 5, .y = 10 },
};
static struct nest2 nest2_designated_both = {
        .s1 = { .x = 5, .y = 10 },
        .s2 = { .x = 5, .y = 10 },
};

static struct {
        int x;
        int y;
```



```
} __attribute__((designated_init))
        anon_positional = { 5, 10 },
        anon_designated = { .x = 5, .y = 10};

static struct s1 s1_array[] = {
        { 5, 10 },
        { .x = 5, .y = 10 },
};

static struct s2 s2_array[] = {
        { 5, 10 },
        { .x = 5, .y = 10 },
};

static struct s1 ret_s1_positional(void)
{
        return ((struct s1){ 5, 10 });
}

static struct s1 ret_s1_designated(void)
{
        return ((struct s1){ .x = 5, .y = 10 });
}

static struct s2 ret_s2_positional(void)
{
        return ((struct s2){ 5, 10 });
}

static struct s2 ret_s2_designated(void)
{
        return ((struct s2){ .x = 5, .y = 10 });
}

static struct nest1 ret_nest1_positional(void)
{
        return ((struct nest1){
                        { 5, 10 },
                        { 5, 10 },
                });
}

static struct nest1 ret_nest1_designated_outer(void)
{
        return ((struct nest1){
                        .s1 = { 5, 10 },
                        .s2 = { 5, 10 },
                });
}

static struct nest1 ret_nest1_designated_inner(void)
{
        return ((struct nest1){
                        { .x = 5, .y = 10 },
                        { .x = 5, .y = 10 },
                });
}

static struct nest1 ret_nest1_designated_both(void)
{
        return ((struct nest1){
                        .s1 = { .x = 5, .y = 10 },
                        .s2 = { .x = 5, .y = 10 },
                });
}
```

```
static struct nest2 ret_nest2_positional(void)
{
        return ((struct nest2){
                        { 5, 10 },
                        { 5, 10 },
                });
}

static struct nest2 ret_nest2_designated_outer(void)
{
        return ((struct nest2){
                        .s1 = { 5, 10 },
                        .s2 = { 5, 10 },
                });
}

static struct nest2 ret_nest2_designated_inner(void)
{
        return ((struct nest2){
                        { .x = 5, .y = 10 },
                        { .x = 5, .y = 10 },
                });
}

static struct nest2 ret_nest2_designated_both(void)
{
        return ((struct nest2){
                        .s1 = { .x = 5, .y = 10 },
                        .s2 = { .x = 5, .y = 10 },
                });
}
/*
 * check-name: designated_init attribute
 *
 * check-error-start
designated-init.c:23:36: warning: in initializer for s2_positional: positional i
designated-init.c:23:39: warning: in initializer for s2_positional: positional i
designated-init.c:27:11: warning: in initializer for s2: positional init of fiel
designated-init.c:27:14: warning: in initializer for s2: positional init of fiel
designated-init.c:31:17: warning: in initializer for s2: positional init of fiel
designated-init.c:31:20: warning: in initializer for s2: positional init of fiel
designated-init.c:42:9: warning: in initializer for nest2_positional: positional
designated-init.c:43:9: warning: in initializer for nest2_positional: positional
designated-init.c:43:11: warning: in initializer for s2: positional init of fiel
designated-init.c:43:14: warning: in initializer for s2: positional init of fiel
designated-init.c:47:17: warning: in initializer for s2: positional init of fiel
designated-init.c:47:20: warning: in initializer for s2: positional init of fiel
designated-init.c:50:9: warning: in initializer for nest2_designated_inner: posi
designated-init.c:51:9: warning: in initializer for nest2_designated_inner: posi
designated-init.c:62:29: warning: in initializer for anon_positional: positional
designated-init.c:62:32: warning: in initializer for anon_positional: positional
designated-init.c:71:11: warning: in initializer for s2: positional init of fiel
designated-init.c:71:14: warning: in initializer for s2: positional init of fiel
designated-init.c:87:30: warning: positional init of field in struct s2, declare
designated-init.c:87:33: warning: positional init of field in struct s2, declare
designated-init.c:99:27: warning: in initializer for s2: positional init of fiel
designated-init.c:99:30: warning: in initializer for s2: positional init of fiel
designated-init.c:107:33: warning: in initializer for s2: positional init of fie
designated-init.c:107:36: warning: in initializer for s2: positional init of fie
designated-init.c:130:25: warning: positional init of field in struct nest2, dec
designated-init.c:131:25: warning: positional init of field in struct nest2, dec
designated-init.c:131:27: warning: in initializer for s2: positional init of fie
designated-init.c:131:30: warning: in initializer for s2: positional init of fie
designated-init.c:139:33: warning: in initializer for s2: positional init of fie
designated-init.c:139:36: warning: in initializer for s2: positional init of fie
designated-init.c:146:25: warning: positional init of field in struct nest2, dec
```



```
designated-init.c:147:25: warning: positional init of field in struct nest2, dec
 * check-error-end
 */
```

```
**********************************************************
     358 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/discarded-label-statement.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 /*
 2  * Verify that the statement following an unused label
 3  * is not discarded with the label.
 4  */

 6 static int bad(int a, int b)
 7 {
 8         int r = 0;

10 start:
11         r += a;
12         r += b;

14         return r;
15 }

17 /*
18  * check-name: discarded-label-statement
19  * check-command: test-linearize $file
20  *
21  * check-output-ignore
22  * check-output-contains: add
23  * check-output-contains: %arg1
24  */
```

```
**********************************************************
     940 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/div.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define INT_MIN         (-__INT_MAX__ - 1)
#define LONG_MIN        (-__LONG_MAX__ - 1)
#define LLONG_MIN       (-__LONG_LONG_MAX__ - 1)

static int xd = 1 / 0;
static int xl = 1L / 0;
static int xll = 1LL / 0;

static int yd = INT_MIN / -1;
static long yl = LONG_MIN / -1;
static long long yll = LLONG_MIN / -1;

static int zd = INT_MIN % -1;
static long zl = LONG_MIN % -1;
static long long zll = LLONG_MIN % -1;

/*
 * check-name: division constants
 *
 * check-error-start
div.c:5:19: warning: division by zero
div.c:6:20: warning: division by zero
div.c:7:22: warning: division by zero
div.c:9:25: warning: constant integer operation overflow
div.c:10:27: warning: constant integer operation overflow
div.c:11:34: warning: constant integer operation overflow
div.c:13:25: warning: constant integer operation overflow
div.c:14:27: warning: constant integer operation overflow
div.c:15:34: warning: constant integer operation overflow
 * check-error-end
 */
```

```
**********************************************************
     179 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/double-semicolon.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern void *memset (void *s, int c, int n);
   2 static void test(void)
   3 {
   4         struct { int foo;; } val;
   5         memset(&val, 0, sizeof(val));
   6 }
   7 /*
   8  * check-name: Double semicolon in struct
   9  */
```

```
**********************************************************
     893 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/dubious-bitwise-with-not.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static unsigned int ok1  = !1 &&  2;
 2 static unsigned int bad1 = !1 &   2;
 3 static unsigned int ok2  = !1 ||  2;
 4 static unsigned int bad2 = !1 |   2;
 5 static unsigned int ok3  =  1 && !2;
 6 static unsigned int bad3 =  1 &  !2;
 7 static unsigned int ok4  =  1 || !2;
 8 static unsigned int bad4 =  1 |  !2;
 9 static unsigned int ok5  = !1 && !2;
10 static unsigned int bad5 = !1 &  !2;
11 static unsigned int ok6  = !1 || !2;
12 static unsigned int bad6 = !1 |  !2;
13 /*
14  * check-name: Dubious bitwise operation on !x
15  *
16  * check-error-start
17 dubious-bitwise-with-not.c:2:31: warning: dubious: !x & y
18 dubious-bitwise-with-not.c:4:31: warning: dubious: !x | y
19 dubious-bitwise-with-not.c:6:31: warning: dubious: x & !y
20 dubious-bitwise-with-not.c:8:31: warning: dubious: x | !y
21 dubious-bitwise-with-not.c:10:31: warning: dubious: !x & !y
22 dubious-bitwise-with-not.c:12:31: warning: dubious: !x | !y
23  * check-error-end
24  */
```

```
**********************************************************
       0 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/empty-file
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
______unchanged_portion_omitted_
```

```
**********************************************************
     333 Fri Dec 21 15:00:50 2018
new/usr/src/tools/smatch/src/validation/endian-big.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#if defined(__LITTLE_ENDIAN__)
#error "__LITTLE_ENDIAN__ defined!"
#endif
#if (__BIG_ENDIAN__ != 1)
#error "__BIG_ENDIAN__ not correctly defined!"
#endif
#if (__BYTE_ORDER__ != __ORDER_BIG_ENDIAN__)
#error "__BYTE_ORDER__ not correctly defined!"
#endif

/*
 * check-name: endian-big.c
 * check-command: sparse -mbig-endian $file
 */
```

```
**********************************************************
     342 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/endian-little.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#if defined(__BIG_ENDIAN__)
#error "__BIG_ENDIAN__ defined!"
#endif
#if (__LITTLE_ENDIAN__ != 1)
#error "__LITTLE_ENDIAN__ not correctly defined!"
#endif
#if (__BYTE_ORDER__ != __ORDER_LITTLE_ENDIAN__)
#error "__BYTE_ORDER__ not correctly defined!"
#endif

/*
 * check-name: endian-little.c
 * check-command: sparse -mlittle-endian $file
 */
```

```
**********************************************************
    2065 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/enum-common.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static enum ENUM_TYPE_A { VALUE_A } var_a;
 2 static enum ENUM_TYPE_B { VALUE_B } var_b;
 3 static enum /* anon. */ { VALUE_C } anon_enum_var;
 4 static int i;

 6 static void take_enum_of_type_a(enum ENUM_TYPE_A arg_enum)
 7 {
 8         (void) arg_enum;
 9 }

11 static void take_int(int arg_int)
12 {
13         (void) arg_int;
14 }

16 static void always_ok(void)
17 {
18         var_a ++;
19         var_a = VALUE_A;
20         var_a = (enum ENUM_TYPE_A) VALUE_B;
21         var_b = (enum ENUM_TYPE_B) i;
22         i = (int) VALUE_A;
23         anon_enum_var = VALUE_C;
24         i = VALUE_C;
25         i = anon_enum_var;
26         i = 7;
27         var_a = (enum ENUM_TYPE_A) 0;
28         anon_enum_var = (__typeof__(anon_enum_var)) 0;
29         anon_enum_var = (__typeof__(anon_enum_var)) VALUE_A;

31         switch (var_a) {
32                 case VALUE_A:
33                 default:
34                         take_enum_of_type_a(var_a);
35                         take_enum_of_type_a(VALUE_A);
36         }

38         switch (anon_enum_var) {
39                 case VALUE_C:
40                 default:
41                         take_int(anon_enum_var);
42         }

44         switch (i) {
45                 case VALUE_C:
46                 default:
47                         take_int(VALUE_C);
48         }
49 }

51 static void trigger_enum_mismatch(void)
52 {
53         switch (var_a) {
54                 case VALUE_B:
55                 case VALUE_C:
56                 default:
57                         take_enum_of_type_a(var_b);
58                         take_enum_of_type_a(VALUE_B);
59         }
```



```
61         switch (anon_enum_var) {
62                 case VALUE_A:
63                 default:
64                         take_enum_of_type_a(anon_enum_var);
65                         take_enum_of_type_a(VALUE_C);
66         }

68         // this has been already working in sparse 0.4.1
69         var_a = var_b;
70         var_b = anon_enum_var;
71         anon_enum_var = var_a;

73         // implemented after sparse 0.4.1
74         var_a = VALUE_B;
75         var_b = VALUE_C;
76         anon_enum_var = VALUE_A;
77 }

79 static void trigger_int_to_enum_conversion(void)
80 {
81         switch (var_a) {
82                 case 0:
83                 default:
84                         take_enum_of_type_a(i);
85                         take_enum_of_type_a(7);
86         }
87         var_a = 0;
88         var_b = i;
89         anon_enum_var = 0;
90         anon_enum_var = i;
91         var_a = (int) VALUE_A;
92         var_a = (int) VALUE_B;
93 }

95 static void trigger_enum_to_int_conversion(void)
96 {
97         i = var_a;
98         i = VALUE_B;
99         switch (i) {
100                case VALUE_A:
101                case VALUE_B:
102                default:
103                        take_int(var_a);
104                        take_int(VALUE_B);
105        }
106 }

108 /*
109  * check-name: enum-common
110  * check-description: common part of the test for -Wenum-mismatch, -Wenum-to-int
111  * check-command: sparse -Wno-enum-mismatch -Wno-int-to-enum $file
112  */
```

```
**********************************************************
    1258 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/enum-from-int.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "enum-common.c"

/*
 * check-name: -Wint-to-enum
 * check-command: sparse -Wno-enum-mismatch $file
 *
 * check-error-start
enum-common.c:84:45: warning: conversion of
enum-common.c:84:45:     int to
enum-common.c:84:45:     int enum ENUM_TYPE_A
enum-common.c:85:45: warning: conversion of
enum-common.c:85:45:     int to
enum-common.c:85:45:     int enum ENUM_TYPE_A
enum-common.c:82:22: warning: conversion of
enum-common.c:82:22:     int to
enum-common.c:82:22:     int enum ENUM_TYPE_A
enum-common.c:87:17: warning: conversion of
enum-common.c:87:17:     int to
enum-common.c:87:17:     int enum ENUM_TYPE_A
enum-common.c:88:17: warning: conversion of
enum-common.c:88:17:     int to
enum-common.c:88:17:     int enum ENUM_TYPE_B
enum-common.c:89:25: warning: conversion of
enum-common.c:89:25:     int to
enum-common.c:89:25:     int enum <noident>
enum-common.c:90:25: warning: conversion of
enum-common.c:90:25:     int to
enum-common.c:90:25:     int enum <noident>
enum-common.c:91:18: warning: conversion of
enum-common.c:91:18:     int to
enum-common.c:91:18:     int enum ENUM_TYPE_A
enum-common.c:92:18: warning: conversion of
enum-common.c:92:18:     int to
enum-common.c:92:18:     int enum ENUM_TYPE_A
 * check-error-end
 */
```

```
**********************************************************
     358 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/enum-mismatch.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
enum ea { A = 0, };
enum eb { B = 1, };


static enum eb foo(enum ea a)
{
        return a;
}

/*
 * check-name: enum-mismatch
 * check-command: sparse -Wenum-mismatch $file
 *
 * check-error-start
enum-mismatch.c:7:16: warning: mixing different enum types
enum-mismatch.c:7:16:     int enum ea  versus
enum-mismatch.c:7:16:     int enum eb
 * check-error-end
 */
```

```
**********************************************************
     936 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/enum-to-int.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "enum-common.c"

/*
 * check-name: -Wenum-to-int
 * check-command: sparse -Wenum-to-int -Wno-enum-mismatch -Wno-int-to-enum $file
 *
 * check-error-start
enum-common.c:97:13: warning: conversion of
enum-common.c:97:13:     int enum ENUM_TYPE_A  to
enum-common.c:97:13:     int
enum-common.c:98:13: warning: conversion of
enum-common.c:98:13:     int enum ENUM_TYPE_B  to
enum-common.c:98:13:     int
enum-common.c:103:34: warning: conversion of
enum-common.c:103:34:     int enum ENUM_TYPE_A  to
enum-common.c:103:34:     int
enum-common.c:104:34: warning: conversion of
enum-common.c:104:34:     int enum ENUM_TYPE_B  to
enum-common.c:104:34:     int
enum-common.c:100:22: warning: conversion of
enum-common.c:100:22:     int enum ENUM_TYPE_A  to
enum-common.c:100:22:     int
enum-common.c:101:22: warning: conversion of
enum-common.c:101:22:     int enum ENUM_TYPE_B  to
enum-common.c:101:22:     int
 * check-error-end
 */
```

```
**********************************************************
     152 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/enum_scope.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
enum {A = 12};

static void f(void)
{
        enum {A = A + 1, B};
        char s[1 - 2 * (B != 14)];
}

/*
 * check-name: enumeration constants' scope [6.2.1p7]
 */
```

```
**********************************************************
    1259 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/escapes.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static int e[] = { '\'', '\"', '\?', '\\',
   2                    '\a', '\b', '\f', '\n', '\r', '\t', '\v',
   3                    '\0', '\012', '\x7890', '\xabcd' };
   4 static char *s = "\'\"\?\\ \a\b\f\n\r\t\v \377\xcafe";

   6 static int bad_e[] = { '\c', '\0123', '\789', '\xdefg' };

   8 static char a_hex[3] = "\x61\x62\x63";
   9 static char b_hex[3] = "\x61\x62\x63\x64";
  10 static char c_hex[3] = "\x61\x62";
  11 static char d_hex[3] = "\x61";

  13 static char a_oct[3] = "\141\142\143";
  14 static char b_oct[3] = "\141\142\143\144";
  15 static char c_oct[3] = "\141\142";
  16 static char d_oct[3] = "\141";
  17 /*
  18  * check-name: Character escape sequences
  19  *
  20  * check-error-start
  21 escapes.c:3:34: warning: hex escape sequence out of range
  22 escapes.c:3:44: warning: hex escape sequence out of range
  23 escapes.c:4:18: warning: hex escape sequence out of range
  24 escapes.c:6:24: warning: unknown escape sequence: '\c'
  25 escapes.c:6:30: warning: multi-character character constant
  26 escapes.c:6:39: warning: multi-character character constant
  27 escapes.c:6:47: warning: hex escape sequence out of range
  28 escapes.c:6:47: warning: multi-character character constant
  29 escapes.c:9:24: warning: too long initializer-string for array of char
  30 escapes.c:14:24: warning: too long initializer-string for array of char
  31  * check-error-end
  32  */
```

**********
249 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/extern-array.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
extern const char *v4l2_type_names[];
const char *v4l2_type_names[] = {
        "test"
};
extern const char *v4l2_type_names[];

static void test(void)
{
        unsigned sz = sizeof(v4l2_type_names);
}
/*
 * check-name: duplicate extern array
 */
```

```
**********************************************************
     391 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/extern-inline.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern __inline__ int f(int);

   3 extern __inline__ int
   4 f(int x)
   5 {
   6         return x;
   7 }

   9 extern int g(int);

  11 extern __inline__ int
  12 g(int x)
  13 {
  14         return x;
  15 }


  18 /*
  19  * check-name: extern inline function
  20  * check-command: sparse $file $file
  21  * check-description: Extern inline function never emits stand alone copy
  22  * of the function. It allows multiple such definitions in different file.
  23  */
```

**********
244 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/external-function-has-definition.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 2 extern void myfunction(void);

 4 extern void
 5 myfunction(void)
 6 {
 7         return;
 8 }

10 /*
11  * check-name: -Wno-external-function-has-definition works
12  * check-command: sparse -Wno-external-function-has-definition
13  * check-error-start
14  * check-error-end
15  */
```

```
**********************************************************
     221 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/field-overlap.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static struct {
        int x;
        struct {
                int z;
                int w;
        } y;
} a = { .y.z = 1, .y.w = 2, };

static struct {int x, y, z;} w[2] = {
        {.x = 1, .y = 2, .z = 3},
        {.x = 1, .y = 2, .z = 3}
};

/*
 * check-name: field overlap
 */
```

```
**********************************************************
    2699 Fri Dec 21 15:00:51 2018
new/usr/src/tools/smatch/src/validation/field-override.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int ref[] = {
        [1] = 3,
        [2] = 3,
        [3] = 3,
        [2] = 2,                /* check-should-warn */
        [1] = 1,                /* check-should-warn */
};

static int foo[] = {
        [1 ... 3] = 3,
};

static int foz[4] = {
        [0 ... 3] = 3,
        [0] = 0,
        [1] = 0,
        [2 ... 3] = 1,
        [2] = 3,                /* check-should-warn */
        [3] = 3,                /* check-should-warn */
};

static int bar[] = {
        [1 ... 3] = 3,
        [1]       = 1,          /* check-should-warn */
        [2]       = 2,          /* check-should-warn */
        [2 ... 4] = 2,          /* check-should-warn */
        [2 ... 3] = 2,          /* check-should-warn */
        [4] = 4,                /* check-should-warn */
        [0] = 0,
        [5] = 5,
};

static int baz[3][3] = {
        [0 ... 2][0 ... 2] = 0,
        [0] = { 0, 0, 0, },     /* check-should-warn */
        [0][0] = 1,             /* check-should-warn */
        [1] = { 0, 0, 0, },     /* check-should-warn */
        [1][0] = 1,             /* check-should-warn */
        [1][1] = 1,             /* check-should-warn */
        [1 ... 2][1 ... 2] = 2,
};


struct s {
        int i;
        int a[2];
};

static struct s s = {
        .a[0] = 0,
        .a[1] = 1,
};

static struct s a[2] = {
        [0].i = 0,
        [1].i = 1,
        [0].a[0] = 2,
        [0].a[1] = 3,
};
```



```
static struct s b[2] = {
        [0 ... 1] = { 0, { 1, 2 }, },
        [0].i = 0,
        [1].i = 1,
        [0].a[0] = 2,
        [0].a[1] = 3,
};

/*
 * check-name: field-override
 * check-command: sparse -Woverride-init -Woverride-init-all $file
 *
 * check-error-start
field-override.c:2:10: warning: Initializer entry defined twice
field-override.c:6:10:   also defined here
field-override.c:3:10: warning: Initializer entry defined twice
field-override.c:5:10:   also defined here
field-override.c:17:10: warning: Initializer entry defined twice
field-override.c:18:10:   also defined here
field-override.c:17:10: warning: Initializer entry defined twice
field-override.c:19:10:   also defined here
field-override.c:23:10: warning: Initializer entry defined twice
field-override.c:24:10:   also defined here
field-override.c:23:10: warning: Initializer entry defined twice
field-override.c:25:10:   also defined here
field-override.c:23:10: warning: Initializer entry defined twice
field-override.c:26:10:   also defined here
field-override.c:26:10: warning: Initializer entry defined twice
field-override.c:27:10:   also defined here
field-override.c:26:10: warning: Initializer entry defined twice
field-override.c:28:10:   also defined here
field-override.c:35:10: warning: Initializer entry defined twice
field-override.c:36:10:   also defined here
field-override.c:37:10: warning: Initializer entry defined twice
field-override.c:38:10:   also defined here
field-override.c:37:10: warning: Initializer entry defined twice
field-override.c:39:10:   also defined here
field-override.c:37:10: warning: Initializer entry defined twice
field-override.c:40:10:   also defined here
 * check-error-end
 */
```

**********
*251 Fri Dec 21 15:00:51 2018*
*new/usr/src/tools/smatch/src/validation/fored_arg.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
 1 /*
 2  * check-name: Forced function argument type.
 3  */

 5 #define __iomem __attribute__((noderef, address_space(2)))
 6 #define __force __attribute__((force))

 8 static void foo(__force void * addr)
 9 {
10 }


13 static void bar(void)
14 {
15         void __iomem  *a;
16         foo(a);
17 }
```

```
**********************************************************
     707 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/foul-bitwise.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned short __attribute__((bitwise))__le16;
static __le16 foo(__le16 a)
{
        return a |= ~a;
}

static int baz(__le16 a)
{
        return ~a == ~a;
}

static int barf(__le16 a)
{
        return a == (a & ~a);
}

static __le16 bar(__le16 a)
{
        return -a;
}

/*
 * check-name: foul bitwise
 * check-error-start
foul-bitwise.c:9:16: warning: restricted __le16 degrades to integer
foul-bitwise.c:9:22: warning: restricted __le16 degrades to integer
foul-bitwise.c:19:16: warning: restricted __le16 degrades to integer
foul-bitwise.c:19:16: warning: incorrect type in return expression (different ba
foul-bitwise.c:19:16:    expected restricted __le16
foul-bitwise.c:19:16:    got int
 * check-error-end
 */
```

```
**********************************************************
     214 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/fp-vs-ptrcast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 float *f01(void* p)
   2 {
   3         return p;
   4 }

   6 /*
   7  * check-name: fp-vs-ptrcast
   8  * check-command: test-linearize -Wno-decl $file
   9  * check-output-ignore
  10  *
  11  * check-output-excludes: fpcast
  12  * check-output-contains: ptrcast
  13  */
```

```
**********************************************************
     137 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/function-pointer-inheritance.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 extern int foo(int f(int, void *));
2
3 int foo(int (*f)(int, void *))
4 {
5     return 0;
6 }
7 /*
8  * check-name: Function pointer inheritance
9  */
```

```
**********************************************************
    2664 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/function-redecl.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#define __user  __attribute__((address_space(1)))
#define NULL    ((void*)0)

int ret_type(void);
void ret_type(void) { }                         /* check-should-fail */


int ret_const(void);
int const ret_const(void) { return 0; }         /* check-should-fail */


void *ret_as(void);
void __user *ret_as(void) { return NULL; }      /* check-should-fail */


void *ret_mod(void);
void const *ret_mod(void) { return NULL; }      /* check-should-fail */


void arg_type(int a);
void arg_type(void *a) { }                      /* check-should-fail */


void arg_const(int a);
void arg_const(const int a) { }                 /* OK */


void arg_as(void *a);
void arg_as(void __user *a) { }                 /* check-should-fail */


void arg_mod(void *);
void arg_mod(void const *a) { }                 /* check-should-fail */


void arg_more_arg(int a);
void arg_more_arg(int a, int b) { }             /* check-should-fail */


void arg_less_arg(int a, int b);
void arg_less_arg(int a) { }                    /* check-should-fail */


void arg_vararg(int a);
void arg_vararg(int a, ...) { }                 /* check-should-fail */

/*
 * check-name: function-redecl
 *
 * check-error-start
function-redecl.c:5:6: error: symbol 'ret_type' redeclared with different type (
function-redecl.c:9:11: error: symbol 'ret_const' redeclared with different type
function-redecl.c:13:13: error: symbol 'ret_as' redeclared with different type (
function-redecl.c:17:12: error: symbol 'ret_mod' redeclared with different type
function-redecl.c:21:6: error: symbol 'arg_type' redeclared with different type
function-redecl.c:29:6: error: symbol 'arg_as' redeclared with different type (o
function-redecl.c:33:6: error: symbol 'arg_mod' redeclared with different type (
function-redecl.c:37:6: error: symbol 'arg_more_arg' redeclared with different t
function-redecl.c:41:6: error: symbol 'arg_less_arg' redeclared with different t
function-redecl.c:45:6: error: symbol 'arg_vararg' redeclared with different typ
```



```
 * check-error-end
 */
```

```
**********************************************************
     304 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/goto-label.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 void foo(void)
 2 {
 3         goto a;
 4 a:
 5 a:
 6         return;
 7 }

 9 void g(void)
10 {
11         goto a;
12 a:
13         return;
14 }

16 void bar(void)
17 {
18         goto neverland;
19 }

21 /*
22  * check-name: goto labels
23  *
24  * check-error-start
25 goto-label.c:5:1: error: label 'a' redefined
26 goto-label.c:18:9: error: label 'neverland' was not declared
27  * check-error-end
28  */
```

```
**********************************************************
     547 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/identifier_list.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 typedef int T;
   2 void f(...);
   3 void g(*);
   4 void h(x,int);
   5 void i_OK(T);
   6 void j(x,T);
   7 /*
   8  * check-name: identifier-list parsing
   9  * check-error-start
  10 identifier_list.c:2:8: warning: variadic functions must have one named argument
  11 identifier_list.c:3:8: error: Expected ) in function declarator
  12 identifier_list.c:3:8: error: got *
  13 identifier_list.c:4:9: error: Expected ) in function declarator
  14 identifier_list.c:4:9: error: got ,
  15 identifier_list.c:6:9: error: Expected ) in function declarator
  16 identifier_list.c:6:9: error: got ,
  17  * check-error-end
  18  */
```

```
**********************************************************
     291 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/implicit-KR-arg-type0.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int foo(a, b)
        int a;
{
        if (b)
                return a;
}

/*
 * check-name: implicit-KR-arg-type
 * check-command: sparse -Wno-decl -Wold-style-definition -Wno-implicit-int $fil
 *
 * check-error-start
implicit-KR-arg-type0.c:2:9: warning: non-ANSI definition of function 'foo'
 * check-error-end
 */
```

```
**********************************************************
     395 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/implicit-ret-type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
fun(void);

foo(void) { return 1; }
static bar(void) { return 1; }

/*
 * check-name: implicit-ret-type.c
 * check-command: sparse -Wno-decl $file
 *
 * check-error-start
implicit-ret-type.c:1:1: warning: 'fun()' has implicit return type
implicit-ret-type.c:3:1: warning: 'foo()' has implicit return type
implicit-ret-type.c:4:8: warning: 'bar()' has implicit return type
 * check-error-end
 */
```

```
**********************************************************
     302 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/implicit-type.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern a;
   2 static b;
   3 c;

   5 /*
   6  * check-name: implicit-type.c
   7  * check-command: sparse -Wno-decl $file
   8  *
   9  * check-error-start
  10 implicit-type.c:1:8: warning: 'a' has implicit type
  11 implicit-type.c:2:8: warning: 'b' has implicit type
  12 implicit-type.c:3:1: warning: 'c' has implicit type
  13  * check-error-end
  14  */
```

```
**********************************************************
     111 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/include-eval.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 /* nothing */

   3 /*
   4  * check-name: include-eval.c
   5  * check-command: sparse -include ./include-eval.inc $file
   6  */
```

```
**********************************************************
     153 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/include-eval.inc
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned long long_t;

inline
static unsigned int ok(void)
{
        return sizeof(long_t);
}

static unsigned int ko(void)
{
        return sizeof(long_t);
}
```

```
**********************************************************
     169 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/infinite-loop0.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void foo(void)
   2 {
   3         int a = a || 0;
   4         if (a) ;
   5 }

   7 /*
   8  * check-name: internal infinite loop (0)
   9  * check-command: sparse -Wno-decl $file
  10  * check-timeout:
  11  */
```

```
**********************************************************
     134 Fri Dec 21 15:00:52 2018
new/usr/src/tools/smatch/src/validation/infinite-loop02.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void foo(void)
   2 {
   3         int a = 1;
   4         while ((a = !a))
   5                 ;
   6 }

   8 /*
   9  * check-name: infinite loop 02
  10  * check-command: sparse -Wno-decl $file
  11  */
```

```
**********************************************************
     203 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/infinite-loop03.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static void foo(int *buf)
   2 {
   3         int a = 1;
   4         int *b;
   5         do {
   6                 if (a)
   7                         b = buf;
   8                 if (a)
   9                         *buf = 0;
  10         } while (!(a = !a));
  11 }

  13 /*
  14  * check-name: infinite loop 03
  15  * check-command: sparse -Wno-decl $file
  16  */
```

**********
416 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/init-char-array.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 /*
 2  * for array of char {<string>} gets special treatment in initializer.
 3  */
 4 static char *s[] = {"aaaaaaaaa"};
 5 static char t[][10] = {"aaaaaaaaa"};
 6 static char u[] = {"aaaaaaaaa"};
 7 static char v[] = "aaaaaaaaa";
 8 static void f(void)
 9 {
10         char x[1/(sizeof(s) == sizeof(char *))];
11         char y[1/(sizeof(u) == 10)];
12         char z[1/(sizeof(v) == 10)];
13         char w[1/(sizeof(t) == 10)];
14 }

15 /*
16  * check-name: char array initializers
17  */
```

```
**********************************************************
     839 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/init-char-array1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * for array of char, ("...") as the initializer is an gcc language
 * extension. check that a parenthesized string initializer is handled
 * correctly and that -Wparen-string warns about it's use.
 */
static const char u[] = ("hello");
static const char v[] = {"hello"};
static const char v1[] = {("hello")};
static const char w[] = "hello";
static const char x[5] = "hello";

static void f(void)
{
        char a[1/(sizeof(u) == 6)];
        char b[1/(sizeof(v) == 6)];
        char c[1/(sizeof(w) == 6)];
        char d[1/(sizeof(x) == 5)];
}
/*
 * check-name: parenthesized string initializer
 * check-command: sparse -Wparen-string $file
 *
 * check-error-start
init-char-array1.c:6:26: warning: array initialized from parenthesized string co
init-char-array1.c:8:28: warning: array initialized from parenthesized string co
 * check-error-end
 */
```

**********
288 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/init_cstring.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
static struct alpha {
  char a[2];
} x = { .a = "ab" };
/*
 * check-name: -Winit-cstring option
 *
 * check-command: sparse -Winit-cstring $file
 * check-error-start
init_cstring.c:3:14: warning: too long initializer-string for array of char(no s
 * check-error-end
 */
```

```
**********************************************************
    1308 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/initializer-entry-defined-twice.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Tests for the "Initializer entry defined twice" warning. */

/* Initializing a struct field twice should trigger the warning. */
struct normal {
        int field1;
        int field2;
};

static struct normal struct_error = {
        .field1 = 0,
        .field1 = 0
};

/* Initializing two different fields of a union should trigger the warning. */
struct has_union {
        int x;
        union {
                int a;
                int b;
        } y;
        int z;
};

static struct has_union union_error = {
        .y = {
                .a = 0,
                .b = 0
        }
};

/* Empty structures can make two fields have the same offset in a struct.
 * Initializing both should not trigger the warning. */
struct empty { };

struct same_offset {
        struct empty field1;
        int field2;
};

static struct same_offset not_an_error = {
        .field1 = { },
        .field2 = 0
};

/*
 * _Bools generally take a whole byte, so ensure that we can initialize
 * them without spewing a warning.
 */
static _Bool boolarray[3] = {
        [0] = 1,
        [1] = 1,
};

/*
 * check-name: Initializer entry defined twice
 *
 * check-error-start
initializer-entry-defined-twice.c:10:10: warning: Initializer entry defined twic
initializer-entry-defined-twice.c:11:10:   also defined here
initializer-entry-defined-twice.c:26:18: warning: Initializer entry defined twic
```



```
initializer-entry-defined-twice.c:27:18:   also defined here
 * check-error-end
 */
```

```
**********************************************************
     203 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/inline_compound_literals.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 struct foo {
 2         int x;
 3 };

 5 static inline void baz(void)
 6 {
 7         (struct foo) { .x = 0 };
 8 }

10 static void barf(void)
11 {
12         baz();
13 }

15 static void foo(void)
16 {
17         baz();
18 }

20 /*
21  * check-name: inline compound literals
22  */
```

```
**********************************************************
    1952 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/int128.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef          __int128        int128_t;
typedef   signed __int128       sint128_t;
typedef unsigned __int128       uint128_t;

typedef __int128 int    badxi;
typedef int __int128    badix;
typedef unsigned unsigned __int128 baduu;
typedef double __int128 baddx;
typedef __int128 double badxd;

int sizeof_int128(void)
{
        return sizeof(__int128);
}

typedef unsigned long long u64;
typedef unsigned long      u32;

u64 foo(u64 a, u64 b, u64 c, u32 s)
{
       unsigned __int128 tmp;

       tmp = (((uint128_t)a) * b) + c;
       return (u64) (tmp >> s);
}

/*
 * check-name: int128
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-contains: ret\\..*\\$16
 * check-output-contains: mulu\\.128
 * check-output-contains: add\\.128
 *
 * check-error-start
int128.c:5:18: error: two or more data types in declaration specifiers
int128.c:5:18: error: Trying to use reserved word 'int' as identifier
int128.c:5:25: error: Expected ; at end of declaration
int128.c:5:25: error: got badxi
int128.c:6:13: error: two or more data types in declaration specifiers
int128.c:6:13: error: Trying to use reserved word '__int128' as identifier
int128.c:6:25: error: Expected ; at end of declaration
int128.c:6:25: error: got badix
int128.c:7:18: error: impossible combination of type specifiers: unsigned unsign
int128.c:7:18: error: Trying to use reserved word 'unsigned' as identifier
int128.c:7:27: error: Expected ; at end of declaration
int128.c:7:27: error: got __int128
int128.c:8:16: error: two or more data types in declaration specifiers
int128.c:8:16: error: Trying to use reserved word '__int128' as identifier
int128.c:8:25: error: Expected ; at end of declaration
int128.c:8:25: error: got baddx
int128.c:9:18: error: two or more data types in declaration specifiers
int128.c:9:18: error: Trying to use reserved word 'double' as identifier
int128.c:9:25: error: Expected ; at end of declaration
int128.c:9:25: error: got badxd
 * check-error-end
 */
```

```
**********************************************************
     115 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/integer-promotions.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int add_char(void)
{
        return (char) 127 + (char) 127 + (char) 2;
}
/*
 * check-name: Integer promotions
 */
```

```
**********************************************************
     266 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/ioc-typecheck.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
extern unsigned int __invalid_size_argument_for_IOC;
static unsigned iocnrs[] = {
        [ 1 ?  0 : __invalid_size_argument_for_IOC ] = 1,
};
/*
 * check-name: integer constant & conditional expression
 * check-known-to-fail
 *
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
     312 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/kill-casts.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 extern void __abort(void);

 3 struct s {
 4         int elem:3;
 5 };

 7 void foo(struct s *x);
 8 void foo(struct s *x)
 9 {
10         if (x->elem == 0) {
11                 if (x->elem != 0 && x->elem != 1)
12                         __abort();
13         }
14 }

16 /*
17  * check-name: kill-casts
18  * check-command: test-linearize $file
19  *
20  * check-output-ignore
21  * check-output-excludes: cast\\.
22  */
```

```
**********************************************************
     227 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/kill-computedgoto.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void foo(int a);
   2 void foo(int a)
   3 {
   4         void *l = &&end + 3;

   6 end:
   7         if (a * 0)
   8                 goto *l;
   9 }

  11 /*
  12  * check-name: kill-computedgoto
  13  * check-command: test-linearize $file
  14  *
  15  * check-output-ignore
  16  * check-output-excludes: add\\.
  17  */
```

```
**********************************************************
     347 Fri Dec 21 15:00:53 2018
new/usr/src/tools/smatch/src/validation/kill-cse.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int foo(int a)
{
        return ((a == 0) + 1) != ((a == 0) + 1);
}

/*
 * check-name: kill-cse
 * check-description:
 *      Verify that instructions removed at CSE are
 *      properly adjust the usage of their operands.
 * check-command: test-linearize -Wno-decl $file
 *
 * check-output-start
foo:
.L0:
        <entry-point>
        ret.32      $0


 * check-output-end
 */
```

```
**********************************************************
     253 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-insert-branch.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void foo(int a)
   2 {
   3         int b = 1;
   4         if (a)
   5                 b++;
   6         if (b)
   7                 ;
   8 }

  10 void bar(int a)
  11 {
  12         if (a ? 1 : 2)
  13                 ;
  14 }

  16 /*
  17  * check-name: kill insert-branch
  18  * check-command: test-linearize -Wno-decl $file
  19  *
  20  * check-output-ignore
  21  * check-output-excludes: select\\.
  22  */
```

```
**********************************************************
     433 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-load.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 int keep(volatile int *p)       { return *p && 0; }
   2 int kill(int *p, int i)         { return *p && 0; }
   3 void ind(volatile int *p,int i) { int v = i++; if (i && 0) p[v]; }


   6 /*
   7  * check-name: kill-load
   8  * check-command: test-linearize -Wno-decl $file
   9  * check-description:
  10  *      Check that loads are optimized away but only
  11  *      when needed:
  12  *      - non-volatile
  13  *      - bb unreachable.
  14  *
  15  * check-output-ignore
  16  * check-output-pattern-1-times: load\\.
  17  */
```

```
**********************************************************
     361 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-phi-node.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 void foo(int a, int *b, unsigned int g);
 2 void foo(int a, int *b, unsigned int g)
 3 {
 4         int d = 0;
 5
 6         if ((!a || *b) && g)
 7                 d = 16;
 8         else
 9                 d = 8;
10 }
11
12 int bar(void);
13 int bar(void)
14 {
15         int i;
16         for (i = 0; i; i--)
17                 ;
18         return 0;
19 }
20
21 /*
22  * check-name: kill-phi-node
23  * check-command: test-linearize $file
24  *
25  * check-output-ignore
26  * check-output-excludes: phisrc\\.
27  */
```

**********
414 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-phi-ttsbb.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 int def(void);
 2 void use(int);

 4 static int foo(int a, int b)
 5 {
 6         int c;

 8         if (a)
 9                 c = 1;
10         else
11                 c = def();

13         if (c)
14                 use(1);
15         else
16                 use(0);
17 }

19 /*
20  * check-name: kill-phi-ttsbb
21  * check-description:
22  *      Verify if OP_PHI usage is adjusted after successful try_to_simplify_bb()
23  * check-command: test-linearize $file
24  * check-output-ignore
25  *
26  * check-output-excludes: phi\\.
27  * check-output-excludes: phisrc\\.
28  */
```

```
**********************************************************
     690 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-phi-ttsbb2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 extern int error(int);

 3 int foo(int perr);
 4 int foo(int perr)
 5 {
 6         int err = 0;
 7         int rc = 0;
 8         int j = 0;
 9         int i = 1;

11         i && j++;

13         i-- && j;

15         i && j--;

17         if (j != 1) {
18                 err = 1;
19                 if (perr)
20                         error(1);
21         }

23         if (err != 0)
24                 rc = 1;

26         return rc;
27 }

29 /*
30  * check-name: kill-phi-ttsbb2
31  * check-description:
32  *      Verify if OP_PHI usage is adjusted after successful try_to_simplify_bb()
33  * check-warning: this test is sensitive to details of code generation
34  *                with proper bb packing (taking care of phi-nodes) it
35  *                will be optimized away and test nothing. You have been warned.
36  * check-command: test-linearize $file
37  * check-output-ignore
38  *
39  * check-output-excludes: VOID
40  */
```

*********************************************************
*260 Fri Dec 21 15:00:54 2018*
*new/usr/src/tools/smatch/src/validation/kill-phisrc.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
int foo(int a, int b)
{
        int r = a + b;

        if (a && 0) {
                int s = r;
                if (b)
                        s = 0;
                (void) s;
        }

        return 0;
}

/*
 * check-name: kill-phisrc
 * check-command: test-linearize -Wno-decl $file
 *
 * check-output-ignore
 * check-output-excludes: add\\.
 */
```

```
**********************************************************
     436 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-pure-call.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 int side(int a);
   2 int pure(int a) __attribute__((pure));

   4 int keep(int a) { return side(a) && 0; }
   5 int kill(int a) { return pure(a) && 0; }

   7 /*
   8  * check-name: kill-pure-call
   9  * check-command: test-linearize -Wno-decl $file
  10  * check-description:
  11  *      See that the call is optimized away but only
  12  *      when the function is "pure".
  13  *
  14  * check-output-ignore
  15  * check-output-contains: call\\..* side
  16  * check-output-excludes: call\\..* pure
  17  */
```

```
**********************************************************
     950 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-replaced-insn.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
// See if the replaced operation is effectively killed or not

static int kill_add(int a, int b)
{
        return (a + b) && 0;
}

static int kill_scast(short a)
{
        return ((int) a) && 0;
}

static int kill_ucast(unsigned char a)
{
        return ((int) a) && 0;
}

static int kill_pcast(int *a)
{
        return ((void*) a) && 0;
}

static int kill_fcast(double a)
{
        return ((int) a) && 0;
}

static int kill_select(int a)
{
        return (a ? 1 : 0) && 0;
}

static int kill_setval(int a)
{
l:
        return &&l && 0;
}

static int kill_load(int *a)
{
        return *a && 0;
}

static int kill_store(int *a)
{
        return (*a = 1) && 0;
}

/*
 * check-name: kill-replaced-insn
 * check-command: test-linearize $file
 *
 * check-output-ignore
 * check-output-excludes: add\\.
 * check-output-excludes: scast\\.
 * check-output-excludes: \\<cast\\.
 * check-output-excludes: ptrcast\\.
 * check-output-excludes: fpcast\\.
 * check-output-excludes: sel\\.
 * check-output-excludes: set\\.
```



```
 */
```

```
**********************************************************
     206 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-rewritten-load.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int foo(int i)
{
        i++;
        if (i && 0)
                i;
        return 0;
}


/*
 * check-name: kill-rewritten-load
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-excludes: add\\.
 */
```

```
**********************************************************
     220 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-select.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 void foo(int x);
 2 void foo(int x)
 3 {
 4         unsigned int ui;

 6         ui = x + 1;
 7         ui = ui ? 0 : 1;
 8 }

10 /*
11  * check-name: kill-select
12  * check-command: test-linearize $file
13  *
14  * check-output-ignore
15  * check-output-excludes: add\\.
16  */
```

```
**********************************************************
     253 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-slice.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct bar {
   2         int x;
   3         int y[2];
   4 };
   5 struct bar bar(void);

   7 int foo(void)
   8 {
   9         int x = bar().x;
  10         return x & 0;
  11 }

  13 /*
  14  * check-name: kill-slice
  15  * check-command: test-linearize -Wno-decl $file
  16  * check-output-ignore
  17  *
  18  * check-output-excludes: slice\\.
  19  */
```

```
**********************************************************
     404 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-store.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void keep(int *p)        { *p = 0; }
   2 void kill(int *p, int i) { if (i && 0) *p = 0; }
   3 void dead(int *p, int i) { int v = i++; if (i && 0) p[v] = 0; }


   6 /*
   7  * check-name: kill-store
   8  * check-command: test-linearize -Wno-decl $file
   9  * check-description:
  10  *      Check that stores are optimized away but only
  11  *      when needed:
  12  *      - bb unreachable.
  13  *
  14  * check-output-ignore
  15  * check-output-pattern-1-times: store\\.
  16  */
```

```
**********************************************************
     536 Fri Dec 21 15:00:54 2018
new/usr/src/tools/smatch/src/validation/kill-unreachable-phi.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
extern char *strcpy (char *__dest, const char *__src);

static void test_menu_iteminfo( void )
{
                int ansi = 1;
                void *init, *string;
                char initA[]="XYZ";
                char stringA[0x80];
                do {
                        if(ansi) {
                                string=stringA;
                                init = initA;
                        }
                        if(ansi)
                                strcpy( string, init );
                } while( !(ansi = !ansi) );
}
/*
 * check-name: kill-unreachable-phi
 * check-description:
 *      In wine source tests/menu.c
 *      Improper killing a phi instruction inside not reachable BB cause
 *      dead loop on sparse.
 *
 * check-output-ignore
 *
 */
```

```
**********************************************************
     235 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/label-asm.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define barrier() __asm__ __volatile__("": : :"memory")

static void f(void)
{
        barrier();
l:
        barrier();
}
/*
 * check-name: Label followed by __asm__
 * check-description: Sparse used to parse the __asm__ as modifying the label.
 */
```

```
**********************************************************
     139 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/label-attr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 static int foo(void)
2 {
3       return 0;
4 rtattr_failure: __attribute__ ((unused))
5       return -1;
6 }
7 /*
8  * check-name: Label attribute
9  */
```

```
**********************************************************
     252 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/label-expr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 int foo(void);
   2 int foo(void)
   3 {
   4         int r;

   6         r = ({ label: 1; });
   7         return r;
   8 }

  10 /*
  11  * check-name: label-expr
  12  * check-command: test-linearize $file
  13  * check-output-ignore
  14  *
  15  * check-output-excludes: ret\\.32\$
  16  * check-output-contains: ret\\.32 *\\$1
  17  */
```

```
**********************************************************
     126 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/label-scope.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int f(int n)
 2 {
 3         __label__ n;
 4 n:      return n;
 5 }
 6 static int g(int n)
 7 {
 8 n:      return n;
 9 }
10 /*
11  * check-name: __label__ scope
12  */
```

**********
515 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/linear/bitfield-init-mask.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 struct bfu {
 2         unsigned int a:11;
 3         unsigned int f:9;
 4         unsigned int z:3;
 5 };

 7 struct bfu bfu_init_00_11(int a)
 8 {
 9         struct bfu bfu = { .a = a, };
10         return bfu;
11 }

13 struct bfu bfu_init_20_23(int a)
14 {
15         struct bfu bfu = { .z = a, };
16         return bfu;
17 }

19 /*
20  * check-name: bitfield initializer mask
21  * check-command: test-linearize -fdump-linearize=only -Wno-decl $file
22  * check-output-ignore
23  *
24  * check-output-contains: and\\..*fffff800\$
25  * check-output-contains: shl\\..* \\$20
26  * check-output-contains: and\\..*ff8fffff\$
27  */
```

```
**********************************************************
    1321 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/linear/bitfield-init-zero.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 struct bfu {
  2         unsigned int a:11;
  3         unsigned int f:9;
  4         unsigned int  :2;
  5         unsigned int z:3;
  6 };

  8 struct bfu bfuu_init(unsigned int a)
  9 {
 10         struct bfu bf = { .f = a, };
 11         return bf;
 12 }

 14 struct bfu bfus_init(int a)
 15 {
 16         struct bfu bf = { .f = a, };
 17         return bf;
 18 }

 20 unsigned int bfu_get0(void)
 21 {
 22         struct bfu bf = { };
 23         return bf.f;
 24 }


 27 struct bfs {
 28         signed int a:11;
 29         signed int f:9;
 30         signed int  :2;
 31         signed int z:3;
 32 };

 34 struct bfs bfsu_init(unsigned int a)
 35 {
 36         struct bfs bf = { .f = a, };
 37         return bf;
 38 }

 40 struct bfs bfss_init(int a)
 41 {
 42         struct bfs bf = { .f = a, };
 43         return bf;
 44 }

 46 int bfs_get0(void)
 47 {
 48         struct bfs bf = { };
 49         return bf.f;
 50 }

 52 /*
 53  * check-name: bitfield implicit init zero
 54  * check-command: test-linearize -Wno-decl $file
 55  *
 56  * check-output-start
 57 bfuu_init:
 58 .L0:
 59         <entry-point>
 60         cast.9      %r2 <- (32) %arg1
```



```
 61         shl.32      %r4 <- %r2, $11
 62         ret.32      %r4


 65 bfus_init:
 66 .L2:
 67         <entry-point>
 68         scast.9     %r10 <- (32) %arg1
 69         shl.32      %r12 <- %r10, $11
 70         ret.32      %r12


 73 bfu_get0:
 74 .L4:
 75         <entry-point>
 76         ret.32      $0


 79 bfsu_init:
 80 .L6:
 81         <entry-point>
 82         cast.9      %r23 <- (32) %arg1
 83         shl.32      %r25 <- %r23, $11
 84         ret.32      %r25


 87 bfss_init:
 88 .L8:
 89         <entry-point>
 90         scast.9     %r31 <- (32) %arg1
 91         shl.32      %r33 <- %r31, $11
 92         ret.32      %r33


 95 bfs_get0:
 96 .L10:
 97         <entry-point>
 98         ret.32      $0


101  * check-output-end
102  */
```

**********
433 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/linear/missing-insn-size.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
int foo(int **a);
int foo(int **a)
{
        return **a;
}

/*
 * check-name: missing instruction's size
 * check-description:
 *      sparse used to have a problem with *all*
 *      double dereferencing due to missing a
 *      call to examine_symbol_type(). The symptom
 *      here is that the inner deref had no type.
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-excludes: load\\s
 * check-output-contains: load\\.
 */
```

```
**********************************************************
     448 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/linear/struct-init-full.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct s {
   2         int a, b, c;
   3 };

   5 struct s s_init_all(int a)
   6 {
   7         struct s s = { .a = a, .b = 42, .c = 123, };
   8         return s;
   9 }

  11 /*
  12  * check-name: struct implicit init zero not needed
  13  * check-command: test-linearize -Wno-decl $file
  14  * check-known-to-fail
  15  *
  16  * check-output-start
  17 s_init_all:
  18 .L4:
  19         <entry-point>
  20         store.32    %arg1 -> 0[s]
  21         store.32    $42 -> 4[s]
  22         store.32    $123 -> 8[s]
  23         load.96     %r8 <- 0[s]
  24         ret.96      %r8


  27  * check-output-end
  28  */
```

```
**********************************************************
     580 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/linear/struct-init-partial.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct s {
        int a, b, c;
};

struct s s_init_first(int a)
{
        struct s s = { .a = a, };
        return s;
}

struct s s_init_third(int a)
{
        struct s s = { .c = a, };
        return s;
}

/*
 * check-name: struct implicit init zero needed
 * check-command: test-linearize -Wno-decl $file
 *
 * check-output-start
s_init_first:
.L0:
        <entry-point>
        store.96    $0 -> 0[s]
        store.32    %arg1 -> 0[s]
        load.96     %r2 <- 0[s]
        ret.96      %r2


s_init_third:
.L2:
        <entry-point>
        store.96    $0 -> 0[s]
        store.32    %arg1 -> 8[s]
        load.96     %r5 <- 0[s]
        ret.96      %r5


 * check-output-end
 */
```

```
**********************************************************
     214 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/local-label.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 void f(unsigned long ip);
 2 static void g(void)
 3 {
 4       if (1) {
 5             f(({ __label__ x; x: (unsigned long)&&x; }));
 6       }
 7       f(({ __label__ x; x: (unsigned long)&&x; }));
 8 }
 9 /*
10  * check-name: Local label
11  */
```

```
**********************************************************
     203 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/logical.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern int a(void);
   2 extern int b(void);
   3 extern int c(void);

   5 static int or(void)
   6 {
   7         return a() || b() || c();
   8 }

  10 static int and(void)
  11 {
  12         return a() && b() && c();
  13 }
  14 /*
  15  * check-name: Logical and/or
  16  */
```

```
**********************************************************
    2023 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/loop-linearization.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 extern int p(int);
  2
  3 static int ffor(void)
  4 {
  5         int i;
  6         for (int i = 0; i < 10; i++) {
  7                 if (!p(i))
  8                         return 0;
  9         }
 10         return 1;
 11 }
 12
 13 static int fwhile(void)
 14 {
 15         int i = 0;
 16         while (i < 10) {
 17                 if (!p(i))
 18                         return 0;
 19                 i++;
 20         }
 21         return 1;
 22 }
 23
 24 static int fdo(void)
 25 {
 26         int i = 0;
 27         do {
 28                 if (!p(i))
 29                         return 0;
 30         } while (i++ < 10);
 31         return 1;
 32 }
 33
 34 /*
 35  * check-name: loop-linearization
 36  * check-command: test-linearize $file
 37  *
 38  * check-output-start
 39 ffor:
 40 .L0:
 41         <entry-point>
 42         phisrc.32   %phi5(i) <- $0
 43         br          .L4
 44
 45 .L4:
 46         phi.32      %r1(i) <- %phi5(i), %phi6(i)
 47         setlt.32    %r2 <- %r1(i), $10
 48         cbr         %r2, .L1, .L3
 49
 50 .L1:
 51         call.32     %r4 <- p, %r1(i)
 52         cbr         %r4, .L2, .L5
 53
 54 .L5:
 55         phisrc.32   %phi1(return) <- $0
 56         br          .L7
 57
 58 .L2:
 59         add.32      %r7 <- %r1(i), $1
 60         phisrc.32   %phi6(i) <- %r7
```



```
 61         br          .L4
 62
 63 .L3:
 64         phisrc.32   %phi2(return) <- $1
 65         br          .L7
 66
 67 .L7:
 68         phi.32      %r5 <- %phi1(return), %phi2(return)
 69         ret.32      %r5
 70
 71
 72 fwhile:
 73 .L8:
 74         <entry-point>
 75         phisrc.32   %phi11(i) <- $0
 76         br          .L12
 77
 78 .L12:
 79         phi.32      %r8(i) <- %phi11(i), %phi12(i)
 80         setlt.32    %r9 <- %r8(i), $10
 81         cbr         %r9, .L9, .L11
 82
 83 .L9:
 84         call.32     %r11 <- p, %r8(i)
 85         cbr         %r11, .L14, .L13
 86
 87 .L13:
 88         phisrc.32   %phi7(return) <- $0
 89         br          .L15
 90
 91 .L14:
 92         add.32      %r14 <- %r8(i), $1
 93         phisrc.32   %phi12(i) <- %r14
 94         br          .L12
 95
 96 .L11:
 97         phisrc.32   %phi8(return) <- $1
 98         br          .L15
 99
100 .L15:
101         phi.32      %r12 <- %phi7(return), %phi8(return)
102         ret.32      %r12
103
104
105 fdo:
106 .L16:
107         <entry-point>
108         phisrc.32   %phi16(i) <- $0
109         br          .L17
110
111 .L17:
112         phi.32      %r15(i) <- %phi16(i), %phi17(i)
113         call.32     %r16 <- p, %r15(i)
114         cbr         %r16, .L18, .L20
115
116 .L20:
117         phisrc.32   %phi13(return) <- $0
118         br          .L22
119
120 .L18:
121         add.32      %r19 <- %r15(i), $1
122         setlt.32    %r20 <- %r15(i), $10
123         phisrc.32   %phi17(i) <- %r19
124         cbr         %r20, .L17, .L19
125
126 .L19:
```

```
127         phisrc.32   %phi14(return) <- $1
128         br          .L22

130 .L22:
131         phi.32      %r17 <- %phi13(return), %phi14(return)
132         ret.32      %r17


135  * check-output-end
136  */
```

```
**********************************************************
     222 Fri Dec 21 15:00:55 2018
new/usr/src/tools/smatch/src/validation/member_of_typeof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static struct foo {int x;} v;
   2 static typeof(v) *p;
   3 static void bar(void)
   4 {
   5         p->x = 0;
   6 }
   7 /*
   8  * check-name: Expansion of typeof when dealing with member of struct
   9  * check-description: Used to expand SYM_TYPEOF too late
  10  */
```

```
**********************************************************
     291 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/memops-volatile.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int foo(volatile int *a, int v)
{
        *a = v;
        return *a;
}

/*
 * check-name: memops-volatile
 * check-command: test-linearize $file
 *
 * check-output-start
foo:
.L0:
        <entry-point>
        store.32    %arg2 -> 0[%arg1]
        load.32     %r5 <- 0[%arg1]
        ret.32      %r5


 * check-output-end
 */
```

```
**********************************************************
     469 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/missing-ident.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int [2];
int *;
int (*);
int ();
int;
struct foo;
union bar {int x; int y;};
struct baz {int x, :3, y:2;};
/*
 * check-name: handling of identifier-less declarations
 *
 * check-error-start
missing-ident.c:1:8: warning: missing identifier in declaration
missing-ident.c:2:6: warning: missing identifier in declaration
missing-ident.c:3:8: warning: missing identifier in declaration
missing-ident.c:4:7: warning: missing identifier in declaration
 * check-error-end
 */
```

```
**********************************************************
     241 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/multi_typedef.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef int T, *P;
static void f(void)
{
        unsigned P = 0;
        unsigned x = P;
}
static void g(void)
{
        int P = 0;
        int x = P;
}
/*
 * check-name: typedefs with many declarators
 * check-description: we didn't recognize P above as a typedef
 */
```

```
**********************************************************
     813 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/nested-declarator.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 typedef int T;
   2 extern void f(int);
   3 static void g(int x)
   4 {
   5         int (T);
   6         T = x;
   7         f(T);
   8 }
   9 static void h(void)
  10 {
  11         static int [2](T)[3];
  12 }
  13 static int [2](*p)[3];
  14 int i(void (void)(*f));
  15 int j(int [2](*));
  16 /*
  17  * check-name: nested declarator vs. parameters
  18  * check-error-start:
  19 nested-declarator.c:11:23: warning: missing identifier in declaration
  20 nested-declarator.c:11:23: error: Expected ; at the end of type declaration
  21 nested-declarator.c:11:23: error: got (
  22 nested-declarator.c:13:15: error: Expected ; at the end of type declaration
  23 nested-declarator.c:13:15: error: got (
  24 nested-declarator.c:14:18: error: Expected ) in function declarator
  25 nested-declarator.c:14:18: error: got (
  26 nested-declarator.c:15:14: error: Expected ) in function declarator
  27 nested-declarator.c:15:14: error: got (
  28  * check-error-end:
  29  */
```

```
**********************************************************
    1242 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/nested-declarator2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef int T;
extern void f1(int);
extern void f2(T);
static void (*f3)(int) = f2;
static void (*f4)(T) = f1;
extern void f5(void (int));
extern void f6(void (T));
static void z(int x)
{
        int (T) = x;
        f5(f2);
        f6(f3);
}
static void f8();
static int (x) = 1;
static void w1(y)
int y;
{
        x = y;
}
static void w2(int ());
static void w3(...);
static void f9(__attribute__((mode(DI))) T);
static void w4(int f(x,y));
static void bad1(__attribute__((mode(DI))) x);
static int (-bad2);
static void [2](*bad3);
/*
 * check-name: more on handling of ( in direct-declarator
 * check-error-start:
nested-declarator2.c:17:1: warning: non-ANSI definition of function 'w1'
nested-declarator2.c:21:21: warning: non-ANSI function declaration of function '
nested-declarator2.c:22:16: warning: variadic functions must have one named argu
nested-declarator2.c:24:21: warning: identifier list not in definition
nested-declarator2.c:25:45: error: don't know how to apply mode to incomplete ty
nested-declarator2.c:26:13: error: Expected ) in nested declarator
nested-declarator2.c:26:13: error: got -
nested-declarator2.c:27:16: error: Expected ; at the end of type declaration
nested-declarator2.c:27:16: error: got (
 * check-error-end:
 */
```

```
**********************************************************
    5134 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/nocast.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __nocast        __attribute__((nocast))
typedef unsigned long __nocast ulong_nc_t;

extern void use_val(ulong_nc_t);
extern void use_ptr(ulong_nc_t *);

/* use address */
static void good_use_address(void)
{
        ulong_nc_t t;

        use_ptr(&t);
}

static ulong_nc_t *good_ret_address(void)
{
        static ulong_nc_t t;

        return &t;
}

static ulong_nc_t good_deref(ulong_nc_t *t)
{
        return *t;
}

/* assign value */
static ulong_nc_t t;
static ulong_nc_t good_assign_self = t;
static unsigned long good_assign_sametype = t;

/* assign pointer */
static ulong_nc_t *good_ptr = &t;
static ulong_nc_t *bad_ptr_to = 1UL;
static unsigned long *bad_ptr_from = &t;

/* arithmetic operation */
static ulong_nc_t good_arith(ulong_nc_t t, unsigned int n)
{
        return t + n;
}

/* implicit cast to other types */
static unsigned long good_ret_samecast(ulong_nc_t t)
{
        return t;
}
static unsigned long long bad_ret_biggercast(ulong_nc_t t)
{
        return t;
}
static long bad_ret_signcast(ulong_nc_t t)
{
        return t;
}
static short bad_ret_smallercast(ulong_nc_t t)
{
        return t;
}
```



```
static void assign_val(ulong_nc_t t)
{
        ulong_nc_t good_c = t;
        unsigned long good_ul = t;
        unsigned long long bad_ull = t;
        long bad_l = t;
        short bad_i = t;
}

static void assign_via_ptr(ulong_nc_t *t)
{
        ulong_nc_t good_c = *t;
        unsigned long good_ul = *t;
        unsigned long long bad_ull = *t;
        long bad_l = *t;
        short bad_i = *t;
}

static void assign_ptr(ulong_nc_t *t)
{
        ulong_nc_t *good_same_type = t;
        unsigned long *bad_mod = t;
        unsigned long long __nocast *bad_size = t;
        short __nocast *bad_i = t;
        long __nocast *bad_l = t;
}

/* implicit cast to nocast */
static void implicit_assign_to(void)
{
        ulong_nc_t t;
        unsigned long ul = 1;
        unsigned short us = 1;
        unsigned long long ull = 1;
        long l = 1;

        t = ul;         /* implicit to nocast from same type: OK? */
        t = us;
        t = ull;
        t = l;
}

static void bad_implicit_arg_to(void)
{
        unsigned long ul = 1;
        unsigned short us = 1;
        unsigned long long ull = 1;
        long l = 1;

        use_val(ul);    /* implicit to nocast from same type: OK? */
        use_val(us);
        use_val(ull);
        use_val(l);
}

/* implicit cast from nocast */
static unsigned long good_implicit_ret_ul(ulong_nc_t t)
{
        return t;       /* implicit to nocast from same type: OK? */
}

static unsigned short bad_implicit_ret_us(ulong_nc_t t)
{
        return t;
}
```

```
127 static unsigned long long bad_implicit_ret_ull(ulong_nc_t t)
128 {
129         return t;
130 }

132 static long bad_implicit_ret_l(ulong_nc_t t)
133 {
134         return t;
135 }

137 /* FIXME: explicit cast: should we complain? */
138 static ulong_nc_t good_samecast(ulong_nc_t v)
139 {
140         return (ulong_nc_t) v;
141 }

143 static ulong_nc_t bad_tocast(unsigned long v)
144 {
145         return (ulong_nc_t) v;
146 }

148 static unsigned long bad_fromcast(ulong_nc_t v)
149 {
150         return (unsigned long) v;
151 }

153 /*
154  * check-name: nocast.c
155  *
156  * check-error-start
157 nocast.c:34:33: warning: incorrect type in initializer (different base types)
158 nocast.c:34:33:    expected unsigned long [nocast] [usertype] *static [toplevel]
159 nocast.c:34:33:    got unsigned long
160 nocast.c:34:33: warning: implicit cast to nocast type
161 nocast.c:35:39: warning: incorrect type in initializer (different modifiers)
162 nocast.c:35:39:    expected unsigned long *static [toplevel] bad_ptr_from
163 nocast.c:35:39:    got unsigned long [nocast] *<noident>
164 nocast.c:35:39: warning: implicit cast from nocast type
165 nocast.c:50:16: warning: implicit cast from nocast type
166 nocast.c:54:16: warning: implicit cast from nocast type
167 nocast.c:58:16: warning: implicit cast from nocast type
168 nocast.c:65:38: warning: implicit cast from nocast type
169 nocast.c:66:22: warning: implicit cast from nocast type
170 nocast.c:67:23: warning: implicit cast from nocast type
171 nocast.c:74:38: warning: implicit cast from nocast type
172 nocast.c:75:22: warning: implicit cast from nocast type
173 nocast.c:76:23: warning: implicit cast from nocast type
174 nocast.c:82:34: warning: incorrect type in initializer (different modifiers)
175 nocast.c:82:34:    expected unsigned long *bad_mod
176 nocast.c:82:34:    got unsigned long [nocast] [usertype] *t
177 nocast.c:82:34: warning: implicit cast from nocast type
178 nocast.c:83:49: warning: incorrect type in initializer (different type sizes)
179 nocast.c:83:49:    expected unsigned long long [nocast] *bad_size
180 nocast.c:83:49:    got unsigned long [nocast] [usertype] *t
181 nocast.c:83:49: warning: implicit cast to/from nocast type
182 nocast.c:84:33: warning: incorrect type in initializer (different type sizes)
183 nocast.c:84:33:    expected short [nocast] *bad_i
184 nocast.c:84:33:    got unsigned long [nocast] [usertype] *t
185 nocast.c:84:33: warning: implicit cast to/from nocast type
186 nocast.c:85:32: warning: implicit cast to/from nocast type
187 nocast.c:98:13: warning: implicit cast to nocast type
188 nocast.c:99:13: warning: implicit cast to nocast type
189 nocast.c:100:13: warning: implicit cast to nocast type
190 nocast.c:111:17: warning: implicit cast to nocast type
191 nocast.c:112:17: warning: implicit cast to nocast type
192 nocast.c:113:17: warning: implicit cast to nocast type
```



```
193 nocast.c:124:16: warning: implicit cast from nocast type
194 nocast.c:129:16: warning: implicit cast from nocast type
195 nocast.c:134:16: warning: implicit cast from nocast type
196  * check-error-end
197  */
```

```
**********************************************************
     788 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/noderef.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 # define __A    __attribute__((noderef))

  3 struct x {
  4         int a;
  5         int b;
  6 };

  8 struct y {
  9         int a[2];
 10 };

 12 static void h(void)
 13 {
 14         char __A *p;
 15         char __A * * q1;
 16         char * __A * q2;
 17         struct x __A *xp;
 18         struct x __A x;
 19         int __A *q;
 20         int __A *r;
 21         struct y __A *py;
 22
 23         q1 = &p;
 24         q2 = &p;        /* This should complain */

 26         r = &*q;
 27         r = q;
 28         r = &*(q+1);    /* This should NOT complain */
 29         r = q+1;

 31         r = &xp->a;     /* This should NOT complain */
 32         r = &xp->b;
 33         r = &(*xp).a;
 34         r = &(*xp).b;

 36         r = &x.a;
 37         r = &x.b;

 39         r = py->a;
 40         r = py->a+1;
 41         r = &py->a[0];
 42 }
 43 /*
 44  * check-name: noderef attribute
 45  *
 46  * check-error-start
 47 noderef.c:24:12: warning: incorrect type in assignment (different modifiers)
 48 noderef.c:24:12:    expected char *[noderef] *q2
 49 noderef.c:24:12:    got char [noderef] **<noident>
 50  * check-error-end
 51  */
```

```
**********************************************************
     191 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/non-pointer-null.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 static void *p = 0;
2 /*
3  * check-name: Using plain integer as NULL pointer
4  *
5  * check-error-start
6 non-pointer-null.c:1:18: warning: Using plain integer as NULL pointer
7  * check-error-end
8  */
```

*********************************************************
*** 167 Fri Dec 21 15:00:56 2018
*new/usr/src/tools/smatch/src/validation/old-initializer-nowarn.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
1 struct s {
2         int i;
3 };

5 static struct s the_s = { i: 1 };
6 /*
7  * check-name: Old initializer with -Wno-old-initializer
8  * check-command: sparse -Wno-old-initializer
9  */
```

```
**********************************************************
     215 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/old-initializer.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct s {
   2         int i;
   3 };

   5 static struct s the_s = { i: 1 };
   6 /*
   7  * check-name: Old initializer
   8  *
   9  * check-error-start
  10 old-initializer.c:5:27: warning: obsolete struct initializer, use C99 syntax
  11  * check-error-end
  12  */
```

```
**********************************************************
     199 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/old-style-definition0.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern int foo(int a, void *b);
   2
   3 int foo(a, b)
   4         int a;
   5         void *b;
   6 {
   7         if (b)
   8                 return a;
   9 }
  10
  11 /*
  12  * check-name: old-stype-definition disabled
  13  * check-command: sparse -Wno-old-style-definition $file
  14  */
```

```
**********************************************************
     314 Fri Dec 21 15:00:56 2018
new/usr/src/tools/smatch/src/validation/old-style-definition1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 extern int foo(int a, void *b);

 3 int foo(a, b)
 4         int a;
 5         void *b;
 6 {
 7         if (b)
 8                 return a;
 9 }

11 /*
12  * check-name: old-stype-definition enabled
13  * check-command: sparse -Wold-style-definition $file
14  *
15  * check-error-start
16 old-style-definition1.c:4:9: warning: non-ANSI definition of function 'foo'
17  * check-error-end
18  */
```

```
**********************************************************
    1337 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/binops-same-args.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int u32;

int ssub(int a) { return a - a; }
u32 usub(u32 a) { return a - a; }

int sdiv(int a) { return a / a; }
u32 udiv(u32 a) { return a / a; }
int smod(int a) { return a % a; }
u32 umod(u32 a) { return a % a; }

int seq(int a) { return a == a; }
int sne(int a) { return a != a; }
int slt(int a) { return a < a; }
int sgt(int a) { return a > a; }
int sle(int a) { return a <= a; }
int sge(int a) { return a >= a; }

u32 ueq(u32 a) { return a == a; }
u32 une(u32 a) { return a != a; }
u32 ult(u32 a) { return a < a; }
u32 ugt(u32 a) { return a > a; }
u32 ule(u32 a) { return a <= a; }
u32 uge(u32 a) { return a >= a; }

u32 xor(u32 a) { return a ^ a; }

u32 ior(u32 a) { return a | a; }
u32 and(u32 a) { return a & a; }

/*
 * check-name: double-unop
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-excludes: sub\\.
 * check-output-contains: divs\\.
 * check-output-contains: divu\\.
 * check-output-contains: mods\\.
 * check-output-contains: modu\\.
 * check-output-excludes: seteq\\.
 * check-output-excludes: setne\\.
 * check-output-excludes: set[gl]t\\.
 * check-output-excludes: set[gl]e\\.
 * check-output-excludes: set[ab]\\.
 * check-output-excludes: set[ab]e\\.
 * check-output-excludes: xor\\.
 * check-output-excludes: or\\.
 * check-output-excludes: and\\.
 */
```

```
**********************************************************
     278 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/bool-context.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define bool _Bool

bool bool_ior(int a, int b) { return a || b; }
bool bool_and(int a, int b) { return a && b; }

/*
 * check-name: bool-context
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-pattern-4-times: setne\\..* %arg[12]
 */
```

```
**********************************************************
     295 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/bool-same-args.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int ior(int a) { return a || a; }
static int and(int a) { return a && a; }

/*
 * check-name: bool-same-args
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-excludes: or-bool\\.
 * check-output-excludes: and-bool\\.
 * check-output-contains: setne\\.
 */
```

**********
582 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/bool-simplify.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
int and_0(int a)
{
        return a && 0;
}

int and_1(int a)
{
        return a && 1;
}

int or_0(int a)
{
        return a || 0;
}

int or_1(int a)
{
        return a || 1;
}

/*
 * check-name: bool-simplify
 * check-command: test-linearize -Wno-decl $file
 *
 * check-output-start
and_0:
.L0:
        <entry-point>
        ret.32      $0


and_1:
.L2:
        <entry-point>
        setne.1     %r8 <- %arg1, $0
        cast.32     %r11 <- (1) %r8
        ret.32      %r11


or_0:
.L4:
        <entry-point>
        setne.1     %r14 <- %arg1, $0
        cast.32     %r17 <- (1) %r14
        ret.32      %r17


or_1:
.L6:
        <entry-point>
        ret.32      $1


 * check-output-end
 */
```

**********
766 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/cse-commutativity.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
static int add(int a, int b) { return (a + b) == (b + a); }
static int mul(int a, int b) { return (a * b) == (b * a); }
static int and(int a, int b) { return (a & b) == (b & a); }
static int ior(int a, int b) { return (a | b) == (b | a); }
static int xor(int a, int b) { return (a ^ b) == (b ^ a); }
static int  eq(int a, int b) { return (a == b) == (b == a); }
static int  ne(int a, int b) { return (a != b) == (b != a); }


/*
 * check-name: cse-commutativity
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-excludes: add\\.
 * check-output-excludes: muls\\.
 * check-output-excludes: and\\.
 * check-output-excludes: or\\.
 * check-output-excludes: xor\\.
 * check-output-excludes: seteq\\.
 * check-output-excludes: setne\\.
 */
```

```
**********************************************************
    1408 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/cse-dual-compare.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 static int eqeq(int a, int b) { return (a == b) == (b == a); }
 2 static int nene(int a, int b) { return (a != b) == (b != a); }

 4 static int ltgt(int a, int b) { return (a <  b) == (b >  a); }
 5 static int lege(int a, int b) { return (a <= b) == (b >= a); }
 6 static int gele(int a, int b) { return (a >= b) == (b <= a); }
 7 static int gtlt(int a, int b) { return (a >  b) == (b <  a); }

 9 static int eneqne(int a, int b) { return (a == b) == !(b != a); }
10 static int enneeq(int a, int b) { return (a != b) == !(b == a); }

12 static int enltle(int a, int b) { return (a <  b) == !(b <= a); }
13 static int enlelt(int a, int b) { return (a <= b) == !(b <  a); }
14 static int engegt(int a, int b) { return (a >= b) == !(b >  a); }
15 static int engtge(int a, int b) { return (a >  b) == !(b >= a); }

17 static int neeqne(int a, int b) { return (a == b) != (b != a); }
18 static int neneeq(int a, int b) { return (a != b) != (b == a); }

20 static int neltle(int a, int b) { return (a <  b) != (b <= a); }
21 static int nelelt(int a, int b) { return (a <= b) != (b <  a); }
22 static int negegt(int a, int b) { return (a >= b) != (b >  a); }
23 static int negtge(int a, int b) { return (a >  b) != (b >= a); }

25 /*
26  * check-name: cse-dual-compare
27  * check-command: test-linearize $file
28  * check-output-ignore
29  * check-known-to-fail
30  *
31  * check-output-excludes: set[gl][et]\\.
32  * check-output-excludes: seteq\\.
33  * check-output-excludes: setne\\.
34  */
```

```
**********************************************************
     351 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/double-unop.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int u32;

u32 unotnot(u32 a) { return ~(~a); }
int snotnot(int a) { return ~(~a); }
u32 unegneg(int a) { return -(-a); }
int snegneg(int a) { return -(-a); }

/*
 * check-name: double-unop
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-excludes: not\\.
 * check-output-excludes: neg\\.
 */
```

```
**********************************************************
     414 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/fpcast-nop.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static  float foof( float a) { return ( float) a; }
static double food(double a) { return (double) a; }
static long double fool(long double a) { return (long double) a; }

/*
 * check-name: fpcast-nop
 * check-description:
 *      Verify that unneeded casts between same-type
 *      floats are also optimized away.
 *
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-excludes: fpcast\\.
 */
```

**********
478 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/muldiv-by-one.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
typedef unsigned int ui;
typedef          int si;

si smul1(si a) {  return a * 1; }
ui umul1(ui a) {  return a * 1; }
si sdiv1(si a) {  return a / 1; }
ui udiv1(ui a) {  return a / 1; }
si smod1(si a) {  return a % 1; }
ui umod1(ui a) {  return a % 1; }

/*
 * check-name: muldiv-by-one
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-excludes: mul[us]\\.
 * check-output-excludes: div[us]\\.
 * check-output-excludes: mod[us]\\.
 */
```

```
**********************************************************
     269 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/muldiv-by-zero.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int ui;
typedef          int si;

si smul0(si a) {  return a * 0; }
ui umul0(ui a) {  return a * 0; }

/*
 * check-name: muldiv-by-zero
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-excludes: mul[us]\\.
 */
```

```
**********************************************************
     480 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/muldiv-minus-one.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef unsigned int u32;

int smulm1(int a) { return a * -1; }
u32 umulm1(u32 a) { return a * (u32) -1; }
int sdivm1(int a) { return a / -1; }
u32 udivm1(u32 a) { return a / (u32) -1; }

/*
 * check-name: muldiv-minus-one
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-excludes: mul[us]\\.
 * check-output-excludes: divs\\.
 * check-output-contains: neg\\.
 * check-output-contains: divu\\.
 * check-output-pattern-3-times: neg\\.
 */
```

**********
685 Fri Dec 21 15:00:57 2018
new/usr/src/tools/smatch/src/validation/optim/setcc-setcc.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
static _Bool blt(int a, int b)  { return  (a <  b); }
static _Bool bnge(int a, int b) { return !(a >= b); }
static _Bool bgt(int a, int b)  { return  (a >  b); }
static _Bool bnle(int a, int b) { return !(a <= b); }
static _Bool ble(int a, int b)  { return  (a <= b); }
static _Bool bngt(int a, int b) { return !(a >  b); }
static _Bool bge(int a, int b)  { return  (a >= b); }
static _Bool bnlt(int a, int b) { return !(a <  b); }

/*
 * check-name: optim/setcc-setcc
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-excludes: set..\\.32
 * check-output-excludes: setne\\.1
 * check-output-excludes: seteq\\.1
 * check-output-contains: set[gt][te]\\.1
 */
```

```
**********************************************************
     353 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/optim/setcc-seteq.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static _Bool beq0(int a)        { return  (a == 0); }
static _Bool bnotneq0(int a)    { return !(a != 0); }
static _Bool bnot(int a)        { return !a; }

/*
 * check-name: optim/setcc-seteq
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-excludes: set..\\.32
 * check-output-excludes: setne\\.1
 * check-output-contains: seteq\\.1
 */
```

```
**********************************************************
     356 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/optim/setcc-setne.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static _Bool bnoteq0(int a)     { return !(a == 0); }
static _Bool bne0(int a)        { return  (a != 0); }
static _Bool bnotnot(int a)     { return !!a; }

/*
 * check-name: optim/setcc-setne
 * check-command: test-linearize $file
 * check-output-ignore
 *
 * check-output-excludes: set..\\.32
 * check-output-excludes: seteq\\.1
 * check-output-contains: setne\\.1
 */
```

```
**********************************************************
     327 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/optim/void-if-convert.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int foo(int a)
{
        if (a)
                return 0;
        else
                return 1;
        return 2;
}

/*
 * check-name: Ignore VOID in if-convert
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-excludes: phisrc\\.
 * check-output-excludes: phi\\.
 * check-output-excludes: VOID
 * check-output-contains: seteq\\.
 */
```

```
**********************************************************
     334 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/outer-scope.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#ifndef FOO
struct st { int len; };
#define FOO
#else
struct st;
static int test(struct st *s);
static int test(struct st *s)
{
        return s->len;
}
#endif
/*
 * check-name: There is no scope boundary between global and file scope
 * check-description: Used to mess scopes with -include
 * check-command: sparse -include $file $file
 */
```

```
**********************************************************
    1726 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/phase2/backslash
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 *      '\\' has a special meaning on phase 2 if and only if it is immediately
 * followed by '\n'.  In any other position it's left alone as any other
 * character.
 *
 * [5.1.1.2(1.2)]:
 *   Each instance of a backslash character (\) immediately followed by
 *   a new-line character is deleted, splicing physical source lines to
 *   form logical source lines.  Only the last backslash on any physical
 *   source line shall be eligible for being part of such a splice.
 *   A source file that is not empty shall end in a new-line character,
 *   which shall not be immediately preceded by a backslash character
 *   before any such splicing takes place.
 *
 * Note that this happens on the phase 2, before we even think of any
 * tokens.  In other words, splicing is ignorant of and transparent for
 * the rest of tokenizer.
 */

#define A(x) #x
#define B(x) A(x)
/* This should result in "\a" */
/* XXX: currently sparse produces "a" */
/* Partially fixed: now it gives "\\a", which is a separate problem */
B(\a)

#define C\
 1
/* This should give 1 */
C

#define D\
1
/* And this should give D, since '\n' is removed and we get no whitespace */
/* XXX: currently sparse produces 1 */
/* Fixed */
D

#define E '\\
a'
/* This should give '\a' - with no warnings issued */
/* XXX: currently sparse complains a lot and ends up producing a */
/* Fixed */
E

/* This should give nothing */
/* XXX: currently sparse produces more junk */
/* Fixed */
// junk \
more junk

/* This should also give nothing */
/* XXX: currently sparse produces / * comment * / */
/* Fixed */
/\
* comment *\
/

/* And this should complain since final newline should not be eaten by '\\' */
/* XXX: currently sparse does not notice */
```



```
/* Fixed */
\
```

```
**********************************************************
     178 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/phase3/comments
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 /*
   2  *  Each comment should be treated as if it had been a single space.
   3  */

   5 /* This should give nothing */
   6 /* XXX: currently sparse produces Y */
   7 /* Fixed */
   8 #define X /*
   9  */ Y
```

```
**********************************************************
      73 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/pragma-once.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 #pragma once
2 #include "pragma-once.c"
3 /*
4  * check-name: #pragma once
5  */
```

```
**********************************************************
     145 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/preprocessor/counter1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
__COUNTER__
__COUNTER__
/*
 * check-name: __COUNTER__ #1
 * check-command: sparse -E $file
 *
 * check-output-start

0
1
 * check-output-end
 */
```

```
**********************************************************
     280 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/preprocessor/counter2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
__FILE__ __COUNTER__
#include <counter2.h>
__FILE__ __COUNTER__
/*
 * check-name: __COUNTER__ #2
 * check-command: sparse -Ipreprocessor -E $file
 *
 * check-output-start

"preprocessor/counter2.c" 0
"preprocessor/counter2.h" 1
"preprocessor/counter2.c" 2
 * check-output-end
 */
```

```
**********************************************************
      21 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/preprocessor/counter2.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 __FILE__ __COUNTER__
```

```
**********************************************************
     266 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/preprocessor/counter3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * check-name: __COUNTER__ #3
 * check-command: sparse -Ipreprocessor -E preprocessor/counter1.c $file
 *
 * check-output-start

0
1
"preprocessor/counter2.c" 0
"preprocessor/counter2.h" 1
"preprocessor/counter2.c" 2
 * check-output-end
 */
#include "counter2.c"
```

**********

171 Fri Dec 21 15:00:58 2018
new/usr/src/tools/smatch/src/validation/preprocessor/dump-macros-empty.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
1 /*
2  * check-name: dump-macros with empty file
3  * check-command: sparse -E -dD empty-file
4  *
5  * check-output-ignore
6 check-output-pattern-1-times: #define __CHECKER__ 1
7  */
```

**********

181 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/dump-macros-multi.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
/*
 * check-name: dump-macros with multiple files
 * check-command: sparse -E -dD empty-file $file
 *
 * check-output-ignore
check-output-pattern-2-times: #define __CHECKER__ 1
 */
```

```
**********************************************************
     386 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/dump-macros.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define ABC abc
#undef ABC

#define DEF def
#undef DEF
#define DEF xyz

#define NYDEF ydef
/*
 * check-name: dump-macros
 * check-command: sparse -E -dD -DIJK=ijk -UNDEF -UNYDEF $file
 *
 * check-output-ignore
check-output-pattern-1-times: #define __CHECKER__ 1
check-output-contains: #define IJK ijk
check-output-contains: #define DEF xyz
check-output-contains: #define NYDEF ydef
 */
```

```
**********************************************************
     471 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/early-escape.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#if 0
"\l"
#endif

/*
 * check-description:
 *      Following the C standard, escape conversion must be
 *      done in phase 5, just after preprocessing and just
 *      before string concatenation. So we're not supposed
 *      to receive a diagnostic for an unknown escape char
 *      for a token which is excluded by the preprocessor.
 * check-name: early-escape
 * check-command: sparse -E $file
 *
 * check-output-start


 * check-output-end
 *
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
     291 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/predef-char-bit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define TEST_BIT(X, T)  if (__ ## X ## _BIT__  != 8 * sizeof(T)) return 1

int test(void)
{
        TEST_BIT(CHAR, char);

        return 0;
}

/*
 * check-name: predefined __<type>_BIT__
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-contains: ret\\..*\\$0
 */
```

**********
348 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/predef-max.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#define TEST_MAX(X, Z)  if (X != ((~ Z) >> 1))  return 1

int test_max(void)
{
        TEST_MAX(__INT_MAX__, 0U);
        TEST_MAX(__LONG_MAX__, 0UL);
        TEST_MAX(__LONG_LONG_MAX__, 0ULL);

        return 0;
}

/*
 * check-name: predefined __<type>_MAX__
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-contains: ret\\..*\\$0
 */
```

```
**********************************************************
     513 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/predef-sizeof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define TEST(X, T)      if (__SIZEOF_ ## X ## __ != sizeof(T))  return 1

int test_sizeof(void)
{
        TEST(SHORT, short);
        TEST(INT, int);
        TEST(LONG, long);
        TEST(LONG_LONG, long long);
        TEST(INT128, __int128);
        TEST(SIZE_T, __SIZE_TYPE__);
        TEST(POINTER, void*);
        TEST(FLOAT, float);
        TEST(DOUBLE, double);
        TEST(LONG_DOUBLE, long double);

        return 0;
}

/*
 * check-name: predefined __SIZEOF_<type>__
 * check-command: test-linearize -Wno-decl $file
 * check-output-ignore
 *
 * check-output-contains: ret\\..*\\$0
 */
```

*********************************************************
*240 Fri Dec 21 15:00:59 2018*
*new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor1.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
#define func(x) x
#define bar func(
#define foo bar foo
foo )
/*
 * check-name: Preprocessor #1
 * check-description: Used to cause infinite recursion.
 * check-command: sparse -E $file
 *
 * check-output-start

foo
 * check-output-end
 */
```

```
**********************************************************
     329 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor10.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* concatenation of 'defi' and 'ned' should result in the same token
 * we would get if we had 'defined' in the input stream.
 */
#define A
#define B defi ## ned
#if B(A)
defined
#else
undefined
#endif
/*
 * check-name: Preprocessor #10
 * check-command: sparse -E $file
 *
 * check-output-start

defined
 * check-output-end
 */
```

```
**********************************************************
    1061 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor11.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#define A(1) x
#define B(x
#define C(x,
#define D(,)
#define E(__VA_ARGS__)
#define F(x+
#define G(x...,
#define H(x...,y)
#define I(...+
#define J(x,y)
/*
 * check-name: Preprocessor #11
 * check-command: sparse -E $file
 *
 * check-output-start


 * check-output-end
 *
 * check-error-start
preprocessor/preprocessor11.c:1:11: error: "1" may not appear in macro parameter
preprocessor/preprocessor11.c:2:11: error: missing ')' in macro parameter list
preprocessor/preprocessor11.c:3:12: error: missing ')' in macro parameter list
preprocessor/preprocessor11.c:4:11: error: parameter name missing
preprocessor/preprocessor11.c:5:11: error: __VA_ARGS__ can only appear in the ex
preprocessor/preprocessor11.c:6:12: error: "+" may not appear in macro parameter
preprocessor/preprocessor11.c:7:12: error: missing ')' in macro parameter list
preprocessor/preprocessor11.c:8:12: error: missing ')' in macro parameter list
preprocessor/preprocessor11.c:9:11: error: missing ')' in macro parameter list
 * check-error-end
 */
```

```
**********************************************************
     206 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor12.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * GNU kludge
 */
#define A(x,...) x,##__VA_ARGS__
A(1)
A(1,2)
A(1,2,3)
/*
 * check-name: Preprocessor #12
 * check-command: sparse -E $file
 *
 * check-output-start

1
1,2
1,2,3
 * check-output-end
 */
```

```
**********************************************************
     444 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor13.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * GNU kludge, corner case
 */
#define A(x,...) x##,##__VA_ARGS__
A(1)
A(1,2)
A(1,2,3)
/*
 * check-name: Preprocessor #13
 * check-command: sparse -E $file
 *
 * check-output-start

1
1,2
1,2,3
 * check-output-end
 *
 * check-error-start
preprocessor/preprocessor13.c:6:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor13.c:7:1: error: '##' failed: concatenation is not a va
 * check-error-end
 */
```

```
**********************************************************
     248 Fri Dec 21 15:00:59 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor14.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 /*
 2  * GNU kludge, another corner case
 3  */
 4 #define A(x,y,...) ,##x##__VA_ARGS__
 5 A(,1)
 6 #define B(x,y,...) x##,##__VA_ARGS__
 7 B(,1)
 8 /*
 9  * check-name: Preprocessor #14
10  * check-command: sparse -E $file
11  *
12  * check-output-start


15  * check-output-end
16  */
```

```
**********************************************************
     220 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor15.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define A defi
#define B ned
#define C(x,y) x##y
#define D(x,y) C(x,y)
#if D(A,B) B
D(1,2)
#endif
/*
 * check-name: Preprocessor #15
 * check-command: sparse -E $file
 *
 * check-output-start

12
 * check-output-end
 */
```

**********
*825 Fri Dec 21 15:01:00 2018*
*new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor16.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
#if 0
/*
From 6.10.1(5):
        Each directive's condition is checked in order.  If it evaluates
        to false (zero), the group it controls is skipped: directives are
        processed only through the name that determines the directive in
        order to keep track of the level of nested conditionals; the rest
        of the directives' preprocessing tokens are ignores, >>as are the
        other preprocessing tokens in the group<<.

In other words, bogus arguments of directives are silently ignored and
so are text lines and non-directives (# <something unknown>).  We *do*
complain about the things like double #else or #elif after #else, since
they hit before we get to the level of groups.
*/

#define 1
#undef 1
#bullshit

#endif
/*
 * check-name: Preprocessor #16
 * check-command: sparse -E $file
 *
 * check-output-start


 * check-output-end
 */
```

```
**********************************************************
     191 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor17.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#if 0
/* these should not warn */
#ifdef (
#endif
#ifndef (
#endif
#endif
/*
 * check-name: Preprocessor #17
 * check-command: sparse -E $file
 * check-output-start


 * check-output-end
 */
```

```
**********************************************************
     367 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor18.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
/* one warning for each, please... */
#define 1
#undef 1
/*
 * check-name: Preprocessor #18
 * check-command: sparse -E $file
 *
 * check-output-start


 * check-output-end
 *
 * check-error-start
preprocessor/preprocessor18.c:2:2: error: expected identifier to 'define'
preprocessor/preprocessor18.c:3:2: error: expected identifier to 'undef'
 * check-error-end
 */
```

```
**********************************************************
     441 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor19.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* got burned by that - freed the new definition in the case when we had
   warned and replaced the old one */
#define A x
#define A y
A
/*
 * check-name: Preprocessor #19
 * check-command: sparse -E $file
 *
 * check-output-start

y
 * check-output-end
 * check-error-start
preprocessor/preprocessor19.c:4:9: warning: preprocessor token A redefined
preprocessor/preprocessor19.c:3:9: this was the original definition
 * check-error-end
 */
```

```
**********************************************************
     208 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define TWO a, b

#define UNARY(x) BINARY(x)
#define BINARY(x, y) x + y

UNARY(TWO)
/*
 * check-name: Preprocessor #2
 * check-command: sparse -E $file
 *
 * check-output-start

a + b
 * check-output-end
 */
```

```
**********************************************************
     199 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor20.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "preprocessor20.h"
#define X
#define Y
#include "preprocessor20.h"
/*
 * check-name: Preprocessor #20
 * check-command: sparse -E $file
 *
 * check-output-start

A
B
 * check-output-end
 */
```

```
**********************************************************
      37 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor20.h
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 #ifdef X
2 B
3 #endif
4 #ifndef Y
5 A
6 #endif
```

```
**********************************************************
     301 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor21.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#if 1
#if
/*
 * check-name: Preprocessor #21
 * check-description: This used to hang Sparse.
 * check-command: sparse -E $file
 *
 * check-output-start


 * check-output-end
 *
 * check-error-start
preprocessor/preprocessor21.c:2:2: error: unterminated preprocessor conditional
 * check-error-end
 */
```

**********
804 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor22.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#define CONFIG_FOO 1

#define define_struct(name, fields...) struct fields name;

define_struct(a, {
#ifdef CONFIG_FOO
  int b;
#elif defined(CONFIG_BAR)
  int c;
#else
  int d;
#endif
});
/*
 * check-name: Preprocessor #22
 *
 * check-description: Directives are not allowed within a macro argument list,
 * although cpp deals with it to treat macro more like C functions.
 *
 * check-command: sparse -E $file
 *
 * check-error-start
preprocessor/preprocessor22.c:6:1: error: directive in argument list
preprocessor/preprocessor22.c:8:1: error: directive in argument list
preprocessor/preprocessor22.c:10:1: error: directive in argument list
preprocessor/preprocessor22.c:12:1: error: directive in argument list
 * check-error-end
 *
 * check-output-start

struct {
int b;
} a;;
 * check-output-end
 */
```

```
**********************************************************
    1387 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor23.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#define H(x,...) ,##x##__VA_ARGS__##,##__VA_ARGS__
H()
H(x)
H(,)
H(x,)
H(,x)
H(x,x)
#define I(x,...) ,##x##__VA_ARGS__
I()
I(x)
I(,)
I(x,)
I(,x)
I(x,x)
#define J(...) ,##__VA_ARGS__
J()
J(x)
/*
 * check-name: Preprocessor #23
 * check-command: sparse -E $file
 *
 * check-output-start

,
,x
,,
,x,
,x,x
,xx,x
,x
,
,x
,x
,xx
,x
 * check-output-end
 *
 * check-error-start
preprocessor/preprocessor23.c:3:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:4:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:5:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:5:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:6:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:6:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:7:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:7:1: error: '##' failed: concatenation is not a va
preprocessor/preprocessor23.c:10:1: error: '##' failed: concatenation is not a v
preprocessor/preprocessor23.c:12:1: error: '##' failed: concatenation is not a v
preprocessor/preprocessor23.c:14:1: error: '##' failed: concatenation is not a v
 * check-error-end
 */
```

```
**********************************************************
     614 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/*
 * Each iteration of the scanning of "SCAN()" re-evaluates the recursive
 * B->A->B expansion.
 *
 * Did I already mention that the C preprocessor language
 * is a perverse thing?
 */

#define LP (

#define A() B LP )
#define B() A LP )

#define SCAN(x) x

A()                     // B ( )
SCAN( A() )             // A ( )
SCAN(SCAN( A() ))       // B ( )
SCAN(SCAN(SCAN( A() ))) // A ( )
/*
 * check-name: Preprocessor #3
 * check-description: Sparse used to get this wrong, outputting A third, not B.
 * check-command: sparse -E $file
 *
 * check-output-start

B ( )
A ( )
B ( )
A ( )
 * check-output-end
 */
```

```
**********************************************************
     244 Fri Dec 21 15:01:00 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define foo bar
#define mac(x) x(foo)

mac(foo)

/*
 * check-name: Preprocessor #4
 * check-description: More examples from the comp.std.c discussion.
 * check-command: sparse -E $file
 *
 * check-output-start

bar(bar)
 * check-output-end
 */
```

```
**********************************************************
     212 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define a a|
#define b(x) x

b(a)
/*
 * check-name: Preprocessor #5
 * check-description: Yet more examples from comp.std.c.
 * check-command: sparse -E $file
 *
 * check-output-start

a|
 * check-output-end
 */
```

```
**********************************************************
     921 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* We used to get '##' wrong for the kernel.
 *
 * It could possibly be argued that the kernel usage is undefined (since the
 * different sides of the '##' are not proper tokens), but we try to do it
 * right anyway.
 *
 * We used to break up the "003d" into two tokens ('003' and 'd') and then put
 * the 'o' marker to mark the token 003 as an octal number, resulting in:
 *
 *      static char __vendorstr_o03 d [ ] __devinitdata = "Lockheed Martin-Marie
 *
 * which didn't work, of course.
 */

#define __devinitdata __attribute__((section(".devinit")))

#define VENDOR( vendor, name ) \
        static char __vendorstr_##vendor[] __devinitdata = name;
VENDOR(003d,"Lockheed Martin-Marietta Corp")

/*
 * check-name: Preprocessor #6
 * check-command: sparse -E $file
 *
 * check-output-start

static char __vendorstr_003d[] __attribute__((section(".devinit"))) = "Lockheed
 * check-output-end
 */
```

```
**********************************************************
     204 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor7.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define A(x) C(B, D
#define D A(1))
#define C(x,y) E(y)
#define E(y) #y
A(2))
/*
 * check-name: Preprocessor #7
 * check-command: sparse -E $file
 *
 * check-output-start

"\"D\""
 * check-output-end
 */
```

```
**********************************************************
     933 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor8.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#define A(x) ## x
#define B(x) x ##
#define C(x) x ## ## ##
#define D(x) x#y
#define E x#y
#define F(x,y) x x##y #x y
#define G a##b
#define H 1##2
#define I(x,y,z) x y z
"A(x)"                  : A(x)
"B(x)"                  : B(x)
"C(x)"                  : C(x)
"D(x)"                  : D(x)
"x#y"                   : E
"ab GH \"G\" 12"        : F(G,H)
"a ## b"                : I(a,##,b)
/*
 * check-name: Preprocessor #8
 * check-command: sparse -E $file
 *
 * check-output-start

"A(x)" : A(x)
"B(x)" : B(x)
"C(x)" : C(x)
"D(x)" : D(x)
"x#y" : x#y
"ab GH \"G\" 12" : ab GH "G" 12
"a ## b" : a ## b
 * check-output-end
 *
 * check-error-start
preprocessor/preprocessor8.c:1:14: error: '##' cannot appear at the ends of macr
preprocessor/preprocessor8.c:2:16: error: '##' cannot appear at the ends of macr
preprocessor/preprocessor8.c:3:22: error: '##' cannot appear at the ends of macr
preprocessor/preprocessor8.c:4:15: error: '#' is not followed by a macro paramet
 * check-error-end
 */
```

```
**********************************************************
     283 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/preprocessor/preprocessor9.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
/* Only # in the input stream marks the beginning of preprocessor command,
 * and here we get it from macro expansion.
 */
#define A # define X 1
A
X
/*
 * check-name: Preprocessor #9
 * check-command: sparse -E $file
 *
 * check-output-start

# define X 1
X
 * check-output-end
 */
```

```
**********************************************************
     295 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/preprocessor/stringify.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define A(x) #x
A('a')
A("a")
A(a)
A(\n)
A('\n')
A("\n")
A('"')
A("a\nb")
A(L"a\nb")
A('\12')
/*
 * check-name: Preprocessor #14
 * check-command: sparse -E $file
 *
 * check-output-start

"'a'"
"\"a\""
"a"
"\n"
"'\\n'"
"\"\\n\""
"'\"'"
"\"a\\nb\""
"L\"a\\nb\""
"'\\12'"
 * check-output-end
 */
```

**********
227 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/preprocessor/wide.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#define A(x) L##x
A('a')
A("bc")
/*
 * check-name: wide char token-pasting
 * check-description: Used to cause infinite recursion.
 * check-command: sparse -E $file
 *
 * check-output-start

L'a'
L"bc"
 * check-output-end
 */
```

```
**********************************************************
     127 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/prototype.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int prototype(void);

/*
 * check-name: Compile skip function prototype
 * check-command: sparsec -c $file -o tmp.o
 */
```

```
**********************************************************
    2030 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/ptr-inherit.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __user          __attribute__((address_space(1)))
#define __noderef       __attribute__((noderef))
#define __bitwise       __attribute__((bitwise))
#define __nocast        __attribute__((nocast))
#define __safe          __attribute__((safe))


/* Should be inherited? */
static void test_const(void)
{
        const int o;
        int *p = &o;                    /* check-should-fail */
}

static void test_volatile(void)
{
        volatile int o;
        int *p = &o;                    /* check-should-fail */
}

static void test_noderef(void)
{
        int __noderef o;
        int *p = &o;                    /* check-should-fail */
}

static void test_bitwise(void)
{
        int __bitwise o;
        int *p = &o;                    /* check-should-fail */
}

static void test_user(void)
{
        int __user o;
        int *p = &o;                    /* check-should-fail */
}

static void test_nocast(void)
{
        int __nocast o;
        int __nocast *p = &o;           /* check-should-pass */
}

/* Should be ignored? */
static void test_static(void)
{
        /* storage is not inherited */
        static int o;
        int *p = &o;                    /* check-should-pass */
}

static void test_tls(void)
{
        /* storage is not inherited */
        static __thread int o;
        int *p = &o;                    /* check-should-pass */
}

/*
```



```
 * check-name: ptr-inherit.c
 *
 * check-error-start
ptr-inherit.c:12:19: warning: incorrect type in initializer (different modifiers
ptr-inherit.c:12:19:    expected int *p
ptr-inherit.c:12:19:    got int const *<noident>
ptr-inherit.c:18:19: warning: incorrect type in initializer (different modifiers
ptr-inherit.c:18:19:    expected int *p
ptr-inherit.c:18:19:    got int volatile *<noident>
ptr-inherit.c:24:19: warning: incorrect type in initializer (different modifiers
ptr-inherit.c:24:19:    expected int *p
ptr-inherit.c:24:19:    got int [noderef] *<noident>
ptr-inherit.c:30:19: warning: incorrect type in initializer (different base type
ptr-inherit.c:30:19:    expected int *p
ptr-inherit.c:30:19:    got restricted int *<noident>
ptr-inherit.c:36:19: warning: incorrect type in initializer (different address s
ptr-inherit.c:36:19:    expected int *p
ptr-inherit.c:36:19:    got int <asn:1>*<noident>
 * check-error-end
 */
```

```
**********************************************************
     219 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/pure-function.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  2 static __attribute__((__pure__)) int pure1(void)
  3 {
  4         int i = 0;
  5         return i;
  6 }

  8 static __attribute__((__pure__)) void *pure2(void)
  9 {
 10     void *i = (void *)0;
 11     return i;
 12 }

 14 /*
 15  * check-name: Pure function attribute
 16  */
```

```
**********************************************************
    7445 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/reserved.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static int (auto);
static int (break);
static int (case);
static int (char);
static int (const);
static int (__const);
static int (__const__);
static int (continue);
static int (default);
static int (do);
static int (double);
static int (else);
static int (enum);
static int (extern);
static int (float);
static int (for);
static int (goto);
static int (if);
static int (inline);
static int (__inline);
static int (__inline__);
static int (int);
static int (long);
static int (register);
static int (restrict);
static int (__restrict);
static int (__restrict__);
static int (return);
static int (short);
static int (signed);
static int (sizeof);
static int (static);
static int (struct);
static int (switch);
static int (typedef);
static int (union);
static int (unsigned);
static int (void);
static int (volatile);
static int (volatile);
static int (__volatile);
static int (__volatile__);
static int (while);

static int (_Alignas);
static int (_Alignof);
static int (_Atomic);
static int (_Bool);
static int (_Complex);
static int (_Generic);
static int (_Imaginary);
static int (_Noreturn);
static int (_Static_assert);
static int (_Thread_local);

// Sparse extensions
static int (__context__);
static int (__range__);
static int (__sizeof_ptr__);
```



```
// GCC extensions
static int (__alignof);
static int (__alignof__);
static int (asm);                       // not reserved!
static int (__asm);
static int (__asm__);
static int (__label__);
static int (__thread);
static int (typeof);
static int (__typeof);
static int (__typeof__);

static int (__int128);
static int (__int128_t);
static int (__uint128_t);

static int (__builtin_ms_va_list);
static int (__builtin_offsetof);
static int (__builtin_types_compatible_p);
static int (__builtin_va_list);

/*
 * check-name: const et.al. are reserved identifiers
 * check-error-start:
reserved.c:1:12: error: Trying to use reserved word 'auto' as identifier
reserved.c:2:12: error: Trying to use reserved word 'break' as identifier
reserved.c:3:12: error: Trying to use reserved word 'case' as identifier
reserved.c:4:12: error: Trying to use reserved word 'char' as identifier
reserved.c:5:12: error: Trying to use reserved word 'const' as identifier
reserved.c:6:12: error: Trying to use reserved word '__const' as identifier
reserved.c:7:12: error: Trying to use reserved word '__const__' as identifier
reserved.c:8:12: error: Trying to use reserved word 'continue' as identifier
reserved.c:9:12: error: Trying to use reserved word 'default' as identifier
reserved.c:10:12: error: Trying to use reserved word 'do' as identifier
reserved.c:11:12: error: Trying to use reserved word 'double' as identifier
reserved.c:12:12: error: Trying to use reserved word 'else' as identifier
reserved.c:13:12: error: Trying to use reserved word 'enum' as identifier
reserved.c:14:12: error: Trying to use reserved word 'extern' as identifier
reserved.c:15:12: error: Trying to use reserved word 'float' as identifier
reserved.c:16:12: error: Trying to use reserved word 'for' as identifier
reserved.c:17:12: error: Trying to use reserved word 'goto' as identifier
reserved.c:18:12: error: Trying to use reserved word 'if' as identifier
reserved.c:19:12: error: Trying to use reserved word 'inline' as identifier
reserved.c:20:12: error: Trying to use reserved word '__inline' as identifier
reserved.c:21:12: error: Trying to use reserved word '__inline__' as identifier
reserved.c:22:12: error: Trying to use reserved word 'int' as identifier
reserved.c:23:12: error: Trying to use reserved word 'long' as identifier
reserved.c:24:12: error: Trying to use reserved word 'register' as identifier
reserved.c:25:12: error: Trying to use reserved word 'restrict' as identifier
reserved.c:26:12: error: Trying to use reserved word '__restrict' as identifier
reserved.c:27:12: error: Trying to use reserved word '__restrict__' as identifie
reserved.c:28:12: error: Trying to use reserved word 'return' as identifier
reserved.c:29:12: error: Trying to use reserved word 'short' as identifier
reserved.c:30:12: error: Trying to use reserved word 'signed' as identifier
reserved.c:31:12: error: Trying to use reserved word 'sizeof' as identifier
reserved.c:32:12: error: Trying to use reserved word 'static' as identifier
reserved.c:33:12: error: Trying to use reserved word 'struct' as identifier
reserved.c:34:12: error: Trying to use reserved word 'switch' as identifier
reserved.c:35:12: error: Trying to use reserved word 'typedef' as identifier
reserved.c:36:12: error: Trying to use reserved word 'union' as identifier
reserved.c:37:12: error: Trying to use reserved word 'unsigned' as identifier
reserved.c:38:12: error: Trying to use reserved word 'void' as identifier
reserved.c:39:12: error: Trying to use reserved word 'volatile' as identifier
reserved.c:40:12: error: Trying to use reserved word 'volatile' as identifier
reserved.c:41:12: error: Trying to use reserved word '__volatile' as identifier
reserved.c:42:12: error: Trying to use reserved word '__volatile__' as identifie
```

```
127 reserved.c:43:12: error: Trying to use reserved word 'while' as identifier
128 reserved.c:45:12: error: Trying to use reserved word '_Alignas' as identifier
129 reserved.c:46:12: error: Trying to use reserved word '_Alignof' as identifier
130 reserved.c:47:12: error: Trying to use reserved word '_Atomic' as identifier
131 reserved.c:48:12: error: Trying to use reserved word '_Bool' as identifier
132 reserved.c:49:12: error: Trying to use reserved word '_Complex' as identifier
133 reserved.c:50:12: error: Trying to use reserved word '_Generic' as identifier
134 reserved.c:51:12: error: Trying to use reserved word '_Imaginary' as identifier
135 reserved.c:52:12: error: Trying to use reserved word '_Noreturn' as identifier
136 reserved.c:53:12: error: Trying to use reserved word '_Static_assert' as identif
137 reserved.c:54:12: error: Trying to use reserved word '_Thread_local' as identifi
138 reserved.c:57:12: error: Trying to use reserved word '__context__' as identifier
139 reserved.c:58:12: error: Trying to use reserved word '__range__' as identifier
140 reserved.c:59:12: error: Trying to use reserved word '__sizeof_ptr__' as identif
141 reserved.c:62:12: error: Trying to use reserved word '__alignof' as identifier
142 reserved.c:63:12: error: Trying to use reserved word '__alignof__' as identifier
143 reserved.c:65:12: error: Trying to use reserved word '__asm' as identifier
144 reserved.c:66:12: error: Trying to use reserved word '__asm__' as identifier
145 reserved.c:67:12: error: Trying to use reserved word '__label__' as identifier
146 reserved.c:68:12: error: Trying to use reserved word '__thread' as identifier
147 reserved.c:69:12: error: Trying to use reserved word 'typeof' as identifier
148 reserved.c:70:12: error: Trying to use reserved word '__typeof' as identifier
149 reserved.c:71:12: error: Trying to use reserved word '__typeof__' as identifier
150 reserved.c:73:12: error: Trying to use reserved word '__int128' as identifier
151 reserved.c:74:12: error: Trying to use reserved word '__int128_t' as identifier
152 reserved.c:75:12: error: Trying to use reserved word '__uint128_t' as identifier
153 reserved.c:77:12: error: Trying to use reserved word '__builtin_ms_va_list' as i
154 reserved.c:78:12: error: Trying to use reserved word '__builtin_offsetof' as ide
155 reserved.c:79:12: error: Trying to use reserved word '__builtin_types_compatible
156 reserved.c:80:12: error: Trying to use reserved word '__builtin_va_list' as iden
157  * check-error-end:
158  */
```

```
**********************************************************
     979 Fri Dec 21 15:01:01 2018
new/usr/src/tools/smatch/src/validation/restrict-array.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __restrict_arr __restrict

struct aiocb64;
struct sigevent;

extern int lio_listio64 (int __mode,
                         struct aiocb64 *__const __list[__restrict_arr],
                         int __nent, struct sigevent *__restrict __sig);

#undef __restrict_arr
#define __restrict_arr __restrict__

struct gaicb;

extern int getaddrinfo_a (int __mode, struct gaicb *__list[__restrict_arr],
                          int __ent, struct sigevent *__restrict __sig);

#undef __restrict_arr
#define __restrict_arr restrict

typedef struct re_pattern_buffer regex_t;
typedef int regoff_t;
typedef struct
{
  regoff_t rm_so;  /* Byte offset from string's start to substring's start.  */
  regoff_t rm_eo;  /* Byte offset from string's start to substring's end.  */
} regmatch_t;
typedef unsigned long int size_t;

extern int regexec (const regex_t *__restrict __preg,
                    const char *__restrict __string, size_t __nmatch,
                    regmatch_t __pmatch[__restrict_arr],
                    int __eflags);

/*
 * check-name: restrict array attribute
 */
```

```
**********************************************************
     188 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/restricted-typeof.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 typedef unsigned __attribute__((bitwise)) A;
   2 static A x;
   3 static __typeof__(x) y;
   4 static A *p = &y;
   5 /*
   6  * check-name: typeof with bitwise types
   7  * check-command: sparse -Wbitwise $file
   8  */
```

**********
*331 Fri Dec 21 15:01:02 2018*
*new/usr/src/tools/smatch/src/validation/sizeof-bool.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
static int a(void)
{
        return sizeof(_Bool);
}
/*
 * check-name: sizeof(_Bool) is valid
 * check-description: sizeof(_Bool) was rejected because _Bool is not an even
 * number of bytes
 * check-command: sparse -Wsizeof-bool $file
 * check-error-start
sizeof-bool.c:3:16: warning: expression using sizeof bool
 * check-error-end
 */
```

```
**********************************************************
     150 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sizeof-compound-postfix.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct foo {int x, y;};
   2 static int a(void)
   3 {
   4         return sizeof (struct foo){0,1}.y;
   5 }
   6 /*
   7  * check-name: Handling of sizeof compound-literal . member
   8  */
```

```
**********************************************************
    1199 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sizeof-void.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define is_constexpr(x) \
        (sizeof(int) == sizeof(*(8 ? ((void *)((long)(x) * 0l)) : (int *)8)))

static int test(void)
{
        unsigned int s = 0, i = 0;
        void *ptr = &i;

        // OK
        s += sizeof i;
        s += sizeof &i;
        s += sizeof ptr;
        s += sizeof &ptr;

        // KO
        s += sizeof(void);
        s += sizeof *ptr;
        s += is_constexpr(ptr++);
        s += is_constexpr((i++, 1));
        s += is_constexpr(sizeof *ptr);
        s += is_constexpr(ptr + 1);
        s += is_constexpr(&ptr + 1);
        s += is_constexpr(*(((char *)&ptr) + 1));

        return s;
}

/*
 * check-name: sizeof(void) is valid
 * check-description: sizeof(void) was rejected because void is an incomplete
 * type
 * check-command: sparse -Wpointer-arith $file
 *
 * check-error-start
sizeof-void.c:16:14: warning: expression using sizeof(void)
sizeof-void.c:17:14: warning: expression using sizeof(void)
sizeof-void.c:18:14: warning: expression using sizeof(void)
sizeof-void.c:19:14: warning: expression using sizeof(void)
sizeof-void.c:20:14: warning: expression using sizeof(void)
sizeof-void.c:21:14: warning: expression using sizeof(void)
sizeof-void.c:22:14: warning: expression using sizeof(void)
sizeof-void.c:23:14: warning: expression using sizeof(void)
 * check-error-end
 */
```

```
**********************************************************
     308 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_WtoA.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void wwww();
   2 void wwwA();

   4 void xxxW (void)
   5 {
   6         wwwA();
   7         www();
   8 }

  10 void DRAW (void)
  11 {
  12         wwwA();
  13 }

  15 void xxxA (void)
  16 {
  17         wwwA();
  18         www();
  19 }


  22 /*
  23  * check-name: Cross calls WtoA
  24  * check-command: smatch -p=wine sm_WtoA.c
  25  *
  26  * check-output-start
  27 sm_WtoA.c:6 xxxW() warn: WtoA call 'wwwA()'
  28  * check-output-end
  29  */
```

```
**********************************************************
     350 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_absolute1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #include "check_debug.h"

   3 char x;
   4 int y;
   5 int func(void)
   6 {
   7         y = x;
   8         __smatch_absolute_min(y);
   9         __smatch_absolute_max(y);
  10 }

  12 /*
  13  * check-name: smatch: absolute #1
  14  * check-command: smatch -I.. sm_absolute1.c
  15  *
  16  * check-output-start
  17 sm_absolute1.c:8 func() absolute min: y = (-128)
  18 sm_absolute1.c:9 func() absolute max: y = 127
  19  * check-output-end
  20  */
```

```
**********************************************************
     851 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_absolute2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

static int my_var;

int x;
int func(int *p)
{
        unsigned int a = -1;

        x = a;
        __smatch_absolute_min(a);
        __smatch_absolute_max(a);
        __smatch_absolute_min(x);
        __smatch_absolute_max(x);
        __smatch_implied(a);
        __smatch_implied(x);
        __smatch_sval_info(a);
        __smatch_sval_info(x);
}
/*
 * check-name: smatch: absolute #2
 * check-command: smatch -I.. sm_absolute2.c
 *
 * check-output-start
sm_absolute2.c:11 func() absolute min: a = u32max
sm_absolute2.c:12 func() absolute max: a = u32max
sm_absolute2.c:13 func() absolute min: x = (-1)
sm_absolute2.c:14 func() absolute max: x = (-1)
sm_absolute2.c:15 func() implied: a = 'u32max'
sm_absolute2.c:16 func() implied: x = '(-1)'
sm_absolute2.c:17 func() implied: a u32 ->value = ffffffff
sm_absolute2.c:18 func() implied: x s32 ->value = ffffffffffffffff
 * check-output-end
 */
```

```
**********************************************************
    1203 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_array_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int a[] = {1, 2, 3, 4};
char *b = "abc";
char c[4];
char d[4] = "";

int x;
static int options_write(void)
{
        int i;
        char *str = b;
        char *str2 = "123";
        char *str3;
        char *str4;
        char *str5;
        unsigned int j = 4;

        str3 = str2;
        str4 = str;
        if (x)
                str5 = "asdf";
        else
                str5 = "aa";

        for (i = 0; i < 4 && frob(); i++)
                ;
        a[i] = 42;
        b[i] = '\0';
        c[i] = '\0';
        str[j] = '\0';
        str2[j] = '\0';
        str3[j] = '\0';
        str4[j] = '\0';
        str5[j] = '\0';
        d[j] = '\0';
}
/*
 * check-name: smatch array check
 * check-command: smatch --spammy sm_array_overflow.c
 *
 * check-output-start
sm_array_overflow.c:26 options_write() error: buffer overflow 'a' 4 <= 4
sm_array_overflow.c:27 options_write() error: buffer overflow 'b' 4 <= 4
sm_array_overflow.c:28 options_write() error: buffer overflow 'c' 4 <= 4
sm_array_overflow.c:29 options_write() error: buffer overflow 'str' 4 <= 4
sm_array_overflow.c:30 options_write() error: buffer overflow 'str2' 4 <= 4
sm_array_overflow.c:31 options_write() error: buffer overflow 'str3' 4 <= 4
sm_array_overflow.c:32 options_write() error: buffer overflow 'str4' 4 <= 4
sm_array_overflow.c:34 options_write() error: buffer overflow 'd' 4 <= 4
 * check-output-end
 */
```

```
**********************************************************
     491 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_array_overflow2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>

#define ARRAY_SIZE(x) (sizeof(x)/sizeof((x)[0]))

int a[] = {1, 2, 3, 4};
int b[] = {
        [3] = 1,
};

int x;
int main(void)
{
        if (x < ARRAY_SIZE(a))
                a[x] = 1;
        if (x < ARRAY_SIZE(b))
                b[x] = 1;
        if (x < ARRAY_SIZE(b))
                b[4] = 1;
        printf("%d\n", ARRAY_SIZE(b));
}
/*
 * check-name: smatch indexed array check
 * check-command: smatch sm_array_overflow2.c
 *
 * check-output-start
sm_array_overflow2.c:18 main() error: buffer overflow 'b' 4 <= 4
 * check-output-end
 */
```

```
**********************************************************
     410 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_array_overflow3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>

#define ARRAY_SIZE(x) (sizeof(x)/sizeof((x)[0]))

int a[] = {1, 2, 3, 4};

int main(void)
{
        int *p;

        for (p = a; p < &a[ARRAY_SIZE(a)]; p++)
                printf("%d\n", *p);
        p = &a[5];
        return 0;
}
/*
 * check-name: smatch array check #3
 * check-command: smatch sm_array_overflow3.c
 *
 * check-output-start
sm_array_overflow3.c:13 main() error: buffer overflow 'a' 4 <= 5
 * check-output-end
 */
```

```
**********************************************************
    1066 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_array_overflow4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>

#define ARRAY_SIZE(x) (sizeof(x)/sizeof((x)[0]))

long long a[] = {1, 2};

int main(char *arg0)
{
        short *s = a;
        short *s2 = (&(a));
        char buf[4], buf2[4];
        int i;

        printf("%d\n", s[1]);
        printf("%d\n", s[2]);
        printf("%d\n", s[3]);
        printf("%d\n", s[4]);
        printf("%d\n", s[5]);
        printf("%d\n", s[6]);
        printf("%d\n", s[7]);
        printf("%d\n", s[8]);
        printf("%d\n", s2[8]);
        printf("%d\n", ((short *)a)[6]);
        printf("%d\n", ((short *)a)[8]);
        strcpy(buf, "1234");
        strcpy(buf2, arg0);

        return 0;
}
/*
 * check-name: smatch overflow check #4
 * check-command: smatch --spammy sm_array_overflow4.c
 *
 * check-output-start
sm_array_overflow4.c:22 main() error: buffer overflow 's' 8 <= 8
sm_array_overflow4.c:23 main() error: buffer overflow 's2' 8 <= 8
sm_array_overflow4.c:25 main() error: buffer overflow 'a' 8 <= 8
sm_array_overflow4.c:26 main() error: strcpy() '"1234"' too large for 'buf' (5 v
sm_array_overflow4.c:27 main() warn: strcpy() 'arg0' of unknown size might be to
 * check-output-end
 */
```

```
**********************************************************
     508 Fri Dec 21 15:01:02 2018
new/usr/src/tools/smatch/src/validation/sm_array_overflow5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

void *kmalloc(int size, int mask);

struct foo {
        int x, y, z;
        int buf[0];
};

int main(void)
{
        struct foo *p;

        p = kmalloc(sizeof(*p) + 100, 0);
        if (!p)
                return -12;
        __smatch_buf_size(p->buf);

        return 0;
}


/*
 * check-name: smatch: overflow check #5
 * check-command: smatch -p=kernel -I.. sm_array_overflow5.c
 *
 * check-output-start
sm_array_overflow5.c:19 main() buf size: 'p->buf' 25 elements, 100 bytes
 * check-output-end
 */
```

```
**********************************************************
     529 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_bitwise1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

unsigned int x;
void test(void)
{
        __smatch_implied(x & 0x1);
        __smatch_implied(x & 0x2);
        __smatch_implied(x & ~(0xffU));
        __smatch_implied(x & ~(0xff));
}

/*
 * check-name: smatch bitwise #1
 * check-command: smatch -I.. sm_bitwise1.c
 *
 * check-output-start
sm_bitwise1.c:6 test() implied: x & 1 = '0-1'
sm_bitwise1.c:7 test() implied: x & 2 = '0,2'
sm_bitwise1.c:8 test() implied: x & ~(255) = '0,256-4294967040'
sm_bitwise1.c:9 test() implied: x & ~(255) = '0-4294967040'
 * check-output-end
 */
```

```
**********************************************************
     381 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_bitwise2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

unsigned int x;
int y;
void test(void)
{
        if (x & 0x1)
                __smatch_implied(x);
        if (y & 0x4)
                __smatch_implied(y);

}

/*
 * check-name: smatch bitwise #2
 * check-command: smatch -I.. sm_bitwise2.c
 *
 * check-output-start
sm_bitwise2.c:8 test() implied: x = '1-u32max'
sm_bitwise2.c:10 test() implied: y = 's32min-(-1),4-s32max'
 * check-output-end
 */
```

```
**********************************************************
     575 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_buf_size1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void func(void)
{
        int a[4];
        char b[4];
        char *c = (char *)a;
        int *d = (int *)b;

        __smatch_buf_size(a);
        __smatch_buf_size(b);
        __smatch_buf_size(c);
        __smatch_buf_size(d);
}

/*
 * check-name: smatch buf size #1
 * check-command: smatch -I.. sm_buf_size1.c
 *
 * check-output-start
sm_buf_size1.c:10 func() buf size: 'a' 4 elements, 16 bytes
sm_buf_size1.c:11 func() buf size: 'b' 4 elements, 4 bytes
sm_buf_size1.c:12 func() buf size: 'c' 16 elements, 16 bytes
sm_buf_size1.c:13 func() buf size: 'd' 1 elements, 4 bytes
 * check-output-end
 */
```

**********
529 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_buf_size2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 #include "check_debug.h"

 3 void *malloc(int);

 5 void func(void)
 6 {
 7         int *a;
 8         short *b;
 9         long long *c;

11         a = malloc(sizeof(int) * 4);
12         b = a;
13         c = b;
14         __smatch_buf_size(a);
15         __smatch_buf_size(b);
16         __smatch_buf_size(c);
17 }

19 /*
20  * check-name: smatch buf size #2
21  * check-command: smatch -I.. sm_buf_size2.c
22  *
23  * check-output-start
24 sm_buf_size2.c:14 func() buf size: 'a' 4 elements, 16 bytes
25 sm_buf_size2.c:15 func() buf size: 'b' 8 elements, 16 bytes
26 sm_buf_size2.c:16 func() buf size: 'c' 2 elements, 16 bytes
27  * check-output-end
28  */
```

```
**********************************************************
     413 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_buf_size3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void *malloc(int);

void func(void)
{
        char *a;

        a = malloc(sizeof(int) * 4);
        __smatch_buf_size(a);
        __smatch_buf_size((int *)a);
}

/*
 * check-name: smatch buf size #3
 * check-command: smatch -I.. sm_buf_size3.c
 *
 * check-output-start
sm_buf_size3.c:10 func() buf size: 'a' 16 elements, 16 bytes
sm_buf_size3.c:11 func() buf size: 'a' 4 elements, 16 bytes
 * check-output-end
 */
```

```
**********************************************************
     500 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_buf_size4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

long long a[] = {1, 2};
int b[] = {3, 4};

int main(char *arg0)
{
        short *s = a;

        __smatch_buf_size(a);
        __smatch_buf_size(b);
        __smatch_buf_size(s);
        return 0;
}
/*
 * check-name: smatch buf size #4
 * check-command: smatch -I.. sm_buf_size4.c
 *
 * check-output-start
sm_buf_size4.c:10 main() buf size: 'a' 2 elements, 16 bytes
sm_buf_size4.c:11 main() buf size: 'b' 2 elements, 8 bytes
sm_buf_size4.c:12 main() buf size: 's' 8 elements, 16 bytes
 * check-output-end
 */
```

*********************************************************
*673 Fri Dec 21 15:01:03 2018*
*new/usr/src/tools/smatch/src/validation/sm_buf_size5.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
#include "check_debug.h"

struct foo {
        char buf[42];
        int x[4];
};

int function(void)
{
        struct foo foo;

        __smatch_buf_size(&foo);
        __smatch_buf_size(&(foo.buf[0]));
        __smatch_buf_size(&foo.x[0]);
        __smatch_buf_size(&foo.x[1]);

        return 0;
}
/*
 * check-name: smatch buf size #5
 * check-command: smatch --spammy -I.. sm_buf_size5.c
 *
 * check-output-start
sm_buf_size5.c:12 function() buf size: '&foo' 1 elements, 60 bytes
sm_buf_size5.c:13 function() buf size: '&(foo.buf[0])' 42 elements, 42 bytes
sm_buf_size5.c:14 function() buf size: '&foo.x[0]' 4 elements, 16 bytes
sm_buf_size5.c:15 function() buf size: '&foo.x[1]' 3 elements, 12 bytes
 * check-output-end
 */
```

*********************************************************
*776 Fri Dec 21 15:01:03 2018*
*new/usr/src/tools/smatch/src/validation/sm_buf_size6.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
 1 #include "check_debug.h"
 2
 3 void *malloc(int size);
 4
 5 int function(void)
 6 {
 7         int *p;
 8         int array[1000];
 9
10         p = malloc(4000);
11
12         __smatch_buf_size(p);
13         __smatch_buf_size(&p[0]);
14         __smatch_buf_size(array);
15         __smatch_buf_size(&array);
16         __smatch_buf_size(&array[0]);
17
18         return 0;
19 }
20 /*
21  * check-name: smatch buf size #6
22  * check-command: smatch --spammy -I.. sm_buf_size6.c
23  *
24  * check-output-start
25 sm_buf_size6.c:12 function() buf size: 'p' 1000 elements, 4000 bytes
26 sm_buf_size6.c:13 function() buf size: '&p[0]' 1000 elements, 4000 bytes
27 sm_buf_size6.c:14 function() buf size: 'array' 1000 elements, 4000 bytes
28 sm_buf_size6.c:15 function() buf size: '&array' 1000 elements, 4000 bytes
29 sm_buf_size6.c:16 function() buf size: '&array[0]' 1000 elements, 4000 bytes
30  * check-output-end
31  */
```

```
**********************************************************
     618 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_buf_size7.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int test(void)
{
        int a[] = { [1] = 2 };
        int b[] = { 1, 2, 3 };
        int c[] = { 0, [0] = 1, 2, 3};
        int d[] = { 0, [3] = 4, 5};

        __smatch_buf_size(a);
        __smatch_buf_size(b);
        __smatch_buf_size(c);
        __smatch_buf_size(d);
}

/*
 * check-name: smatch buf size #7
 * check-command: smatch -I.. sm_buf_size7.c
 *
 * check-output-start
sm_buf_size7.c:10 test() buf size: 'a' 2 elements, 8 bytes
sm_buf_size7.c:11 test() buf size: 'b' 3 elements, 12 bytes
sm_buf_size7.c:12 test() buf size: 'c' 3 elements, 12 bytes
sm_buf_size7.c:13 test() buf size: 'd' 5 elements, 20 bytes
 * check-output-end
 */
```

```
**********************************************************
     991 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_buf_size8.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdlib.h>
#include "check_debug.h"

struct foo {
        int x, y, z;
        int count;
        char msg[0];
};

struct bar {
        int x, y, z;
        int count;
        char msg[1];
};

struct outer1 {
        int x, y, z;
        struct foo foo;
};

struct outer2 {
        int x, y, z;
        struct bar bar;
};

int test(void)
{
        struct foo *p;
        struct bar *q;
        struct outer1 *a;
        struct outer2 *b;

        p = malloc(sizeof(*p) + 100);
        __smatch_buf_size(p->msg);

        q = malloc(sizeof(*q) + 100);
        __smatch_buf_size(q->msg);

        a = malloc(sizeof(*a) + 100);
        __smatch_buf_size(a->foo);

        b = malloc(sizeof(*b) + 100);
        __smatch_buf_size(b->bar);
}

/*
 * check-name: smatch buf size #8
 * check-command: smatch -I.. sm_buf_size8.c
 *
 * check-output-start
sm_buf_size8.c:34 test() buf size: 'p->msg' 100 elements, 100 bytes
sm_buf_size8.c:37 test() buf size: 'q->msg' 101 elements, 101 bytes
sm_buf_size8.c:40 test() buf size: 'a->foo' 0 elements, 116 bytes
sm_buf_size8.c:43 test() buf size: 'b->bar' 0 elements, 120 bytes
 * check-output-end
 */
```

```
**********************************************************
     834 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_casts.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void frob(void);
   2
   3 char c;
   4 static int options_write(void)
   5 {
   6         char a;
   7         unsigned char b;
   8
   9         a = (char)0xff;
  10         a = 0xff;
  11         (char)b = 0xff;
  12         b = 0xff;
  13         if (c > -400)
  14                 frob();
  15         if (c < -400)
  16                 frob();
  17         if (400 > c)
  18                 frob();
  19         if (-400 > c)
  20                 frob();
  21         b = -12;
  22
  23 }
  24 /*
  25  * check-name: smatch cast handling
  26  * check-command: smatch sm_casts.c
  27  *
  28  * check-output-start
  29 sm_casts.c:13 options_write() warn: always true condition '(c > -400) => ((-128)
  30 sm_casts.c:15 options_write() warn: impossible condition '(c < -400) => ((-128)-
  31 sm_casts.c:17 options_write() warn: always true condition '(400 > c) => (400 > (
  32 sm_casts.c:19 options_write() warn: impossible condition '(-400 > c) => ((-400)
  33 sm_casts.c:21 options_write() warn: assigning (-12) to unsigned variable 'b'
  34  * check-output-end
  35  */
```

```
**********************************************************
     658 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_casts2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>

unsigned int frob();

unsigned char *a;
unsigned int *b;
int *c;
unsigned char *****d;
int main(void)
{

        if (*a == (unsigned int)-1)
                frob();
        if (*b == (unsigned int)-1)
                frob();
        if (*c == (unsigned int)-1)
                frob();
        if (*d == (unsigned int)-1)
                frob();
        if (*d == -1)
                frob();
        if (*****d == (unsigned int)-1)
                frob();
        return 0;
}
/*
 * check-name: smatch casts pointers
 * check-command: smatch sm_casts2.c
 *
 * check-output-start
sm_casts2.c:12 main() warn: impossible condition '(*a == -1) => (0-255 == u32max
sm_casts2.c:22 main() warn: impossible condition '(*****d == -1) => (0-255 == u3
 * check-output-end
 */
```

```
**********************************************************
     334 Fri Dec 21 15:01:03 2018
new/usr/src/tools/smatch/src/validation/sm_casts3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

unsigned int a;
int b;
int func(void)
{
        b = a;
        __smatch_implied(a);
        __smatch_implied(b);
}

/*
 * check-name: smatch: casts #3
 * check-command: smatch -I.. sm_casts3.c
 *
 * check-output-start
sm_casts3.c:8 func() implied: a = ''
sm_casts3.c:9 func() implied: b = 's32min-s32max'
 * check-output-end
 */
```

```
**********************************************************
    1002 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_casts4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

long long l;
unsigned long long ul;
int i;
unsigned int ui;
signed char c;
unsigned char uc;

int main(void)
{
        int idx;

        if (c < -2)
                return 1;
        if (uc < -2)
                return 1;
        if (i < -2)
                return 1;
        if (ui < -2)
                return 1;
        if (l < -2)
                return 1;
        if (ul < -2)
                return 1;

        __smatch_implied(l);
        __smatch_implied(ul);
        __smatch_implied(i);
        __smatch_implied(ui);
        __smatch_implied(c);
        __smatch_implied(uc);

        return 0;
}


/*
 * check-name: smatch: casts #4
 * check-command: smatch -I.. sm_casts4.c
 *
 * check-output-start
sm_casts4.c:18 main() warn: impossible condition '(uc < -2) => (0-255 < (-2))'
sm_casts4.c:29 main() implied: l = '(-2)-s64max'
sm_casts4.c:30 main() implied: ul = '18446744073709551614-u64max'
sm_casts4.c:31 main() implied: i = '(-2)-s32max'
sm_casts4.c:32 main() implied: ui = '4294967294-u32max'
sm_casts4.c:33 main() implied: c = '(-2)-127'
sm_casts4.c:34 main() implied: uc = ''
 * check-output-end
 */
```

```
**********************************************************
     929 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_casts5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

long long l;
long long ul;
int i;
int ui;
signed char c;
char uc;

int main(void)
{
        int idx;

        if (c < -2)
                return 1;
        if (uc < (unsigned int)-2)
                return 1;
        if (i < -2)
                return 1;
        if (ui < (unsigned int)-2)
                return 1;
        if (l < -2)
                return 1;
        if (ul < (unsigned int)-2)
                return 1;

        __smatch_implied(l);
        __smatch_implied(ul);
        __smatch_implied(i);
        __smatch_implied(ui);
        __smatch_implied(c);
        __smatch_implied(uc);

        return 0;
}


/*
 * check-name: smatch: casts #5
 * check-command: smatch -I.. sm_casts5.c
 *
 * check-output-start
sm_casts5.c:29 main() implied: l = '(-2)-s64max'
sm_casts5.c:30 main() implied: ul = '4294967294-s64max'
sm_casts5.c:31 main() implied: i = '(-2)-s32max'
sm_casts5.c:32 main() implied: ui = '(-2)-(-1)'
sm_casts5.c:33 main() implied: c = '(-2)-127'
sm_casts5.c:34 main() implied: uc = '(-2)-(-1)'
 * check-output-end
 */
```

```
**********************************************************
     924 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_casts6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

long long l;
long long ul;
int i;
int ui;
signed char c;
char uc;

int main(void)
{
        int idx;

        if (c < 2)
                return 1;
        if (uc < (unsigned int)2)
                return 1;
        if (i < 2)
                return 1;
        if (ui < (unsigned int)2)
                return 1;
        if (l < 2)
                return 1;
        if (ul < (unsigned int)2)
                return 1;

        __smatch_implied(l);
        __smatch_implied(ul);
        __smatch_implied(i);
        __smatch_implied(ui);
        __smatch_implied(c);
        __smatch_implied(uc);

        return 0;
}


/*
 * check-name: smatch: casts #6
 * check-command: smatch -I.. sm_casts6.c
 *
 * check-output-start
sm_casts6.c:29 main() implied: l = '2-s64max'
sm_casts6.c:30 main() implied: ul = '2-s64max'
sm_casts6.c:31 main() implied: i = '2-s32max'
sm_casts6.c:32 main() implied: ui = 's32min-(-1),2-s32max'
sm_casts6.c:33 main() implied: c = '2-127'
sm_casts6.c:34 main() implied: uc = '(-128)-(-1),2-127'
 * check-output-end
 */
```

```
**********************************************************
     331 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_casts7.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

int a;
int x;

int main(void)
{
        a = (unsigned short)x;
        __smatch_implied(a);

        return 0;
}


/*
 * check-name: smatch: casts #7
 * check-command: smatch -I.. sm_casts7.c
 *
 * check-output-start
sm_casts7.c:11 main() implied: a = '0-u16max'
 * check-output-end
 */
```

```
**********************************************************
     552 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_check_kunmap.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void something();

   3 int kmap(int p);
   4 int kunmap(int p);
   5 int kmap_atomic(int p);
   6 int kunmap_atomic(int p);

   8 int page;
   9 int x;
  10 int y;
  11 int z;

  13 void func(void)
  14 {
  15         x = kmap(page);
  16         kunmap(page);
  17         kunmap(x);
  18         y = kmap_atomic(z);
  19         kunmap_atomic(y);
  20         kunmap_atomic(z);
  21 }
  22 /*
  23  * check-name: smatch check kunmap
  24  * check-command: smatch -p=kernel sm_check_kunmap.c
  25  *
  26  * check-output-start
  27 sm_check_kunmap.c:17 func() warn: passing the wrong variable to kunmap()
  28 sm_check_kunmap.c:20 func() warn: passing the wrong variable to kmap_atomic()
  29  * check-output-end
  30  */
```

```
**********************************************************
     481 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_chunk1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

static void perf_calculate_period(unsigned long nsec, unsigned long count)
{
        if (nsec + count > 64)
                return;

        __smatch_implied(nsec + count);
        nsec = 100;
        __smatch_implied(nsec + count);
}


/*
 * check-name: smatch chunk #1
 * check-command: smatch -I.. sm_chunk1.c
 *
 * check-output-start
sm_chunk1.c:8 perf_calculate_period() implied: nsec + count = '0-64'
sm_chunk1.c:10 perf_calculate_period() implied: nsec + count = ''
 * check-output-end
 */
```

```
**********************************************************
     768 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_chunk2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void initialize(void *p);

int main(int x)
{
        unsigned int aaa[10];
        int y, z;

        initialize(&aaa);
        initialize(&y);
        initialize(&z);

        if (aaa[5] > 3)
                return 0;
        aaa[0] = 42;
        __smatch_implied(aaa[0]);
        __smatch_implied(aaa[5]);
        aaa[y] = 10;
        __smatch_implied(aaa[5]);
        if (aaa[z] > 4)
                return 0;
        __smatch_implied(aaa[z]);
        z = 3;
        __smatch_implied(aaa[z]);

        return 0;
}

/*
 * check-name: smatch chunk #2
 * check-command: smatch -I.. sm_chunk2.c
 *
 * check-output-start
sm_chunk2.c:17 main() implied: aaa[0] = '42'
sm_chunk2.c:18 main() implied: aaa[5] = '0-3'
sm_chunk2.c:20 main() implied: aaa[5] = '0-u32max'
sm_chunk2.c:23 main() implied: aaa[z] = '0-4'
sm_chunk2.c:25 main() implied: aaa[z] = '0-u32max'
 * check-output-end
 */
```

```
**********************************************************
     712 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_compare.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b, c;

static int frob(void)
{
        if (a > 5)
                return;
        if (b > 5)
                return;
        if (c != 5)
                return;

        if (a == 10)
                __smatch_value("a");
        if (b != 10)
                __smatch_value("b");
        if (c != 5)
                __smatch_value("c");
        if (5 != c)
                __smatch_value("c");

        __smatch_value("a");
        __smatch_value("b");
        __smatch_value("c");
}

/*
 * check-name: Smatch Comparison
 * check-command: smatch -I.. sm_compare.c
 *
 * check-output-start
sm_compare.c:15 frob() a = empty
sm_compare.c:17 frob() b = s32min-5
sm_compare.c:19 frob() c = empty
sm_compare.c:21 frob() c = empty
sm_compare.c:23 frob() a = s32min-5
sm_compare.c:24 frob() b = s32min-5
sm_compare.c:25 frob() c = 5
 * check-output-end
 */
```

```
**********************************************************
     317 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_compare10.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b, c;
static int options_write(void)
{
        if (c <= b)
                return;
        if (a >= b)
                return;
        __smatch_compare(a, c);
}

/*
 * check-name: smatch compare #10
 * check-command: smatch -I.. sm_compare10.c
 *
 * check-output-start
sm_compare10.c:10 options_write() a < c
 * check-output-end
 */
```

```
**********************************************************
     383 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_compare11.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int frob();

static int options_write(void)
{
        int a = frob();
        int b = frob();
        int c = frob();
        int d = frob();

        a = d;
        if (a > b + c) {
                a = b + c;
        }
        __smatch_compare(a, d);
}

/*
 * check-name: smatch compare #11
 * check-command: smatch -I.. sm_compare11.c
 *
 * check-output-start
sm_compare11.c:16 options_write() a <= d
 * check-output-end
 */
```

```
**********************************************************
     792 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_compare12.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

#define min_t(type, x, y) ({                    \
        type __min1 = (x);                      \
        type __min2 = (y);                      \
        __min1 < __min2 ? __min1: __min2; })

int frob();

static int options_write(void)
{
        int a = frob();
        int b = frob();
        int c = frob();
        int d = frob();

        a = min_t(int, b + c, d);
        __smatch_compare(a, d);
        __smatch_compare(a, b + c);
        b++;
        __smatch_compare(a, b + c);
        a++;  /* argh...  really one increment should mean a <= b + c */
        a++;
        __smatch_compare(a, b + c);

}

/*
 * check-name: smatch compare #12
 * check-command: smatch -I.. sm_compare12.c
 *
 * check-output-start
sm_compare12.c:18 options_write() a <= d
sm_compare12.c:19 options_write() a <= b + c
sm_compare12.c:21 options_write() a < b + c
sm_compare12.c:24 options_write() a <none> b + c
 * check-output-end
 */
```

**********
469 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_compare13.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 #include <stdio.h>
 2 #include <string.h>
 3 #include "check_debug.h"

 5 int cmp_x(int x, int y)
 6 {
 7         if (x < y) {
 8                 __smatch_compare(x, y);
 9                 return -1;
10         }
11         if (x == y) {
12                 __smatch_compare(x, y);
13                 return 0;
14         }
15         __smatch_compare(x, y);
16         return 1;
17 }

19 /*
20  * check-name: smatch compare #13
21  * check-command: smatch -I.. sm_compare13.c
22  *
23  * check-output-start
24 sm_compare13.c:8 cmp_x() x < y
25 sm_compare13.c:12 cmp_x() x == y
26 sm_compare13.c:15 cmp_x() x > y
27  * check-output-end
28  */
```

**********
491 Fri Dec 21 15:01:04 2018
new/usr/src/tools/smatch/src/validation/sm_compare14.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

int cmp_x(int x, int y)
{
        if (x < y)
                return -1;
        if (x == y)
                return 0;
        return 1;
}

int x, y;
int test(void)
{
        if (cmp_x(x, 4) < 0) {
                __smatch_implied(x);
        } else
                __smatch_implied(x);
}
/*
 * check-name: smatch compare #14
 * check-command: smatch -I.. sm_compare14.c
 *
 * check-output-start
sm_compare14.c:18 test() implied: x = 's32min-3'
sm_compare14.c:20 test() implied: x = '4-s32max'
 * check-output-end
 */
```

```
**********************************************************
     371 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare15.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

int __fswab(int x)
{
        return x;
}

int a;
int cmp_x(int x, int y)
{
        if (__fswab(a) > 5)
                return;
        __smatch_implied(a);
}


/*
 * check-name: smatch compare #15
 * check-command: smatch -I.. sm_compare15.c
 *
 * check-output-start
sm_compare15.c:15 cmp_x() implied: a = 's32min-5'
 * check-output-end
 */
```

```
**********************************************************
     373 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare16.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

int return_x(int x)
{
        return x;
}

int a;
int cmp_x(int x, int y)
{
        if (a > return_x(5))
                return;
        __smatch_implied(a);
}


/*
 * check-name: smatch compare #16
 * check-command: smatch -I.. sm_compare16.c
 *
 * check-output-start
sm_compare16.c:15 cmp_x() implied: a = 's32min-5'
 * check-output-end
 */
```

```
**********************************************************
     421 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare17.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int frob();

int xxx;
static int options_write(void)
{
        int a = -1;
        int found = 0;

        if (xxx < 0)
                return;
        while (frob()) {
                if (++a == xxx) {
                        found = 1;
                        break;
                }
        }
        if (!found)
                __smatch_compare(a, xxx);
}

/*
 * check-name: smatch compare #17
 * check-command: smatch -I.. sm_compare17.c
 *
 * check-output-start
sm_compare17.c:20 options_write() a < xxx
 * check-output-end
 */
```

```
**********************************************************
     664 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b, c;

int main(void)
{
        if (a < 4)
                return 1;
        if (a > 10)
                return 2;
        __smatch_value("a");

        if (b < 3)
                return 3;
        if (b > 15)
                return 4;
        __smatch_value("b");

        if (b > a) {
                __smatch_value("a");
                __smatch_value("b");
        } else {
                __smatch_value("a");
                __smatch_value("b");
        }
        return 5;
}

/*
 * check-name: Smatch Comparison #2
 * check-command: smatch -I.. sm_compare2.c
 *
 * check-output-start
sm_compare2.c:11 main() a = 4-10
sm_compare2.c:17 main() b = 3-15
sm_compare2.c:20 main() a = 4-10
sm_compare2.c:21 main() b = 5-15
sm_compare2.c:23 main() a = 4-10
sm_compare2.c:24 main() b = 3-10
 * check-output-end
 */
```

```
**********************************************************
     988 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

int a, b, c, d;
int e, f, g;
int main(void)
{
        if (b > 1000000000)
                return 0;

        if (a >= b)
                return 1;
        if (a < 0 || b < 0)
                return 1;
        c = b - a;
        __smatch_implied(c);
        __smatch_compare(b, c);

        if (e < 0 || e > b)
                return;
        if (f <= 0 || f > b)
                return;
        g = e + f;

        __smatch_implied(g);
        __smatch_implied(e);
        __smatch_compare(g, e);
        __smatch_compare(e, g);
        __smatch_implied(g - e);
        __smatch_implied(g - f);

        return 0;
}

/*
 * check-name: Smatch compare #3
 * check-command: smatch -I.. sm_compare3.c
 *
 * check-output-start
sm_compare3.c:17 main() implied: c = '1-1000000000'
sm_compare3.c:18 main() b <= c
sm_compare3.c:26 main() implied: g = '1-2000000000'
sm_compare3.c:27 main() implied: e = '0-1000000000'
sm_compare3.c:28 main() g > e
sm_compare3.c:29 main() e < g
sm_compare3.c:30 main() implied: g - e = '1-2000000000'
sm_compare3.c:31 main() implied: g - f = '0-1999999999'
 * check-output-end
 */
```

```
**********************************************************
     632 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int some_func();

int a, b, c, d;

void func (void)
{
        d = some_func();

        if (a + 3 > 100)
                return;
        __smatch_implied(a);
        if (3 + b > 100)
                return;
        __smatch_implied(b);
        if (c - 3 > 100)
                return;
        __smatch_implied(c);
        if (3 - d > 100)
                return;
        __smatch_implied(d);
}

/*
 * check-name: Smatch compare #4
 * check-command: smatch -I.. sm_compare4.c
 *
 * check-output-start
sm_compare4.c:13 func() implied: a = 's32min-97'
sm_compare4.c:16 func() implied: b = 's32min-97'
sm_compare4.c:19 func() implied: c = 's32min-103'
sm_compare4.c:22 func() implied: d = 's32min-s32max'
 * check-output-end
 */
```

```
**********************************************************
     546 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b, c, d;
void func(void)
{
        a = b + 3;
        c = d - 3;

        if (a > 10)
                return;
        __smatch_implied(a);
        __smatch_implied(b);
        if (10 > c)
                return;
        __smatch_implied(c);
        __smatch_implied(d);
}

/*
 * check-name: Smatch compare #5
 * check-command: smatch -I.. sm_compare5.c
 *
 * check-output-start
sm_compare5.c:11 func() implied: a = 's32min-10'
sm_compare5.c:12 func() implied: b = 's32min-7'
sm_compare5.c:15 func() implied: c = '10-s32max'
sm_compare5.c:16 func() implied: d = '13-s32max'
 * check-output-end
 */
```

```
**********************************************************
     307 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int returns_less(int x)
{
        int y;

        if (x > 10)
                y = 10;
        else
                y = x;

        __smatch_compare(x, y);
        return y;
}

/*
 * check-name: smatch compare #6
 * check-command: smatch -I.. sm_compare6.c
 *
 * check-output-start
sm_compare6.c:12 returns_less() x >= y
 * check-output-end
 */
```

```
**********************************************************
     402 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare7.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b, c, e, f, g;
static int options_write(void)
{
        if (b >= c)
                return;
        a = c;
        __smatch_compare(a, b);
        if (f >= e)
                return;
        g = f;
        __smatch_compare(g, e);
}

/*
 * check-name: smatch compare #7
 * check-command: smatch -I.. sm_compare7.c
 *
 * check-output-start
sm_compare7.c:9 options_write() a > b
sm_compare7.c:13 options_write() g < e
 * check-output-end
 */
```

```
**********************************************************
     279 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare8.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #include "check_debug.h"

   3 void *a, *b;
   4 static int options_write(void)
   5 {
   6         a = b + 1;
   7         __smatch_compare(a, b);
   8 }

  10 /*
  11  * check-name: smatch compare #8
  12  * check-command: smatch -I.. sm_compare8.c
  13  *
  14  * check-output-start
  15 sm_compare8.c:7 options_write() a > b
  16  * check-output-end
  17  */
```

**********
279 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compare9.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#include "check_debug.h"

void *a, *b;
static int options_write(void)
{
        a = b / 2;
        __smatch_compare(a, b);
}

/*
 * check-name: smatch compare #9
 * check-command: smatch -I.. sm_compare9.c
 *
 * check-output-start
sm_compare9.c:7 options_write() a < b
 * check-output-end
 */
```

```
**********************************************************
     444 Fri Dec 21 15:01:05 2018
new/usr/src/tools/smatch/src/validation/sm_compound_condition.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct ture {
   2         int a;
   3 };

   5 struct ture *a;
   6 struct ture *b;

   8 void func (void)
   9 {
  10         if (!a && !(a = returns_nonnull()))
  11                 return;
  12         a->a = 1;

  14         if (b || (b = returns_nonnull())) {
  15                 b->a  = 1;
  16                 return;
  17         }
  18         b->a = 1;
  19 }
  20 /*
  21  * check-name: Compound Conditions
  22  * check-command: smatch sm_compound_condition.c
  23  *
  24  * check-output-start
  25 sm_compound_condition.c:18 func() error: we previously assumed 'b' could be null
  26  * check-output-end
  27  */
```

```
**********************************************************
    1556 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_compound_conditions2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int aaa;
int x, y, z;

void func (void)
{
        aaa = 0;
        if (y)
                aaa = 1;
        if (x)
                aaa = 2;

        if (x) {
                __smatch_value("aaa");
                if (y)
                        __smatch_value("aaa");
                else
                        __smatch_value("aaa");
        }
        if (!x) {
                __smatch_value("aaa");
                if (y)
                        __smatch_value("aaa");
                else
                        __smatch_value("aaa");
        }
        if (y) {
                __smatch_value("aaa");
                if (x)
                        __smatch_value("aaa");
                else
                        __smatch_value("aaa");
        }
        if (!y) {
                __smatch_value("aaa");
                if (x)
                        __smatch_value("aaa");
                else
                        __smatch_value("aaa");
        }
        if (x && y)
                __smatch_value("aaa");
        if (x || y)
                __smatch_value("aaa");
        else
                __smatch_value("aaa");
        if (!x && !y)
                __smatch_value("aaa");
}
/*
 * check-name: Compound Conditions #2
 * check-command: smatch -I.. sm_compound_conditions2.c
 *
 * check-output-start
sm_compound_conditions2.c:15 func() aaa = 2
sm_compound_conditions2.c:17 func() aaa = 2
sm_compound_conditions2.c:19 func() aaa = 2
sm_compound_conditions2.c:22 func() aaa = 0-1
sm_compound_conditions2.c:24 func() aaa = 1
```



```
sm_compound_conditions2.c:26 func() aaa = 0
sm_compound_conditions2.c:29 func() aaa = 1-2
sm_compound_conditions2.c:31 func() aaa = 2
sm_compound_conditions2.c:33 func() aaa = 1
sm_compound_conditions2.c:36 func() aaa = 0,2
sm_compound_conditions2.c:38 func() aaa = 2
sm_compound_conditions2.c:40 func() aaa = 0
sm_compound_conditions2.c:43 func() aaa = 2
sm_compound_conditions2.c:45 func() aaa = 1-2
sm_compound_conditions2.c:47 func() aaa = 0
sm_compound_conditions2.c:49 func() aaa = 0
 * check-output-end
 */
```

```
**********************************************************
     638 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_compound_conditions3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int aaa;

void func (void)
{
        if (aaa > 0 && aaa < 100) {
                __smatch_value("aaa");
        } else {
                __smatch_value("aaa");
        }
        if (aaa > 0 && aaa < 100 && aaa < 10) {
                __smatch_value("aaa");
        } else {
                if (aaa != 42)
                        __smatch_value("aaa");
        }
}
/*
 * check-name: Compound Conditions #3
 * check-command: smatch -I.. sm_compound_conditions3.c
 *
 * check-output-start
sm_compound_conditions3.c:8 func() aaa = 1-99
sm_compound_conditions3.c:10 func() aaa = s32min-0,100-s32max
sm_compound_conditions3.c:13 func() aaa = 1-9
sm_compound_conditions3.c:16 func() aaa = s32min-0,10-41,43-s32max
 * check-output-end
 */
```

```
**********************************************************
     626 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_deref_check_deref.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct ture {
   2         int a;
   3 };
   4 struct cont {
   5         struct ture *x;
   6 };

   8 struct ture *x;
   9 struct ture **px;
  10 struct cont *y;
  11 void func (void)
  12 {
  13         int *a = &(x->a);
  14         int *b = &x->a;
  15         int *c = &(y->x->a);
  16         int *d = &((*px)->a);

  18         if (x)
  19                 frob();
  20         if (px)
  21                 frob();
  22         if (y->x)
  23                 frob();
  24         if (y)
  25                 frob();

  27         return;
  28 }
  29 /*
  30  * check-name: Dereferencing before check
  31  * check-command: smatch sm_deref_check_deref.c
  32  *
  33  * check-output-start
  34 sm_deref_check_deref.c:20 func() warn: variable dereferenced before check 'px' (
  35 sm_deref_check_deref.c:24 func() warn: variable dereferenced before check 'y' (s
  36  * check-output-end
  37  */
```

```
**********************************************************
     378 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_dev_hold.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void dev_hold(int *x);
   2
   3 void dev_put(int *x){}
   4
   5 extern int y,z;
   6 int *x;
   7
   8 int func (void)
   9 {
  10         dev_hold(x);
  11         if (y) {
  12                 dev_put(x);
  13                 return -1;
  14         }
  15         if (z) {
  16                 return -1;
  17         }
  18         return 0;
  19 }
  20 /*
  21  * check-name: dev_hold() check
  22  * check-command: smatch --project=kernel sm_dev_hold.c
  23  *
  24  * check-output-start
  25 sm_dev_hold.c:16 func() warn: 'x' held on error path.
  26  * check-output-end
  27  */
```

```
**********************************************************
     287 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_double_free1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdlib.h>

void func (void)
{
        void *x;

        x = malloc(42);

        free(x);
        free(x);

        return 0;
}
/*
 * check-name: double free test #1
 * check-command: smatch sm_double_free1.c
 *
 * check-output-start
sm_double_free1.c:10 func() error: double free of 'x'
 * check-output-end
 */
```

**********
354 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_double_free2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 #include <stdlib.h>
 2
 3 struct ture {
 4         int a;
 5 };
 6
 7 void func (void)
 8 {
 9         void *x;
10
11         x = malloc(sizeof(struct ture));
12         x->a = 1;
13
14         if (x->a)
15                 free(x);
16
17         free(x);
18
19         return 0;
20 }
21 /*
22  * check-name: double free test #2
23  * check-command: smatch sm_double_free2.c
24  *
25  * check-output-start
26 sm_double_free2.c:17 func() error: double free of 'x'
27  * check-output-end
28  */
```

```
**********************************************************
     355 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_efault.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int clear_user();

int func(int *p)
{
        int ret;

        ret = clear_user();
        if (ret)
                return ret;
        return 0;
}
/*
 * check-name: smatch return -EFAULT
 * check-command: smatch -p=kernel -I.. sm_efault.c
 *
 * check-output-start
sm_efault.c:11 func() warn: maybe return -EFAULT instead of the bytes remaining?
 * check-output-end
 */
```

```
**********************************************************
     797 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_equiv1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#include "check_debug.h"

int *something();

int *one;
int *two;
int func(void)
{
        one = something();
        two = one;

        if (two == 1) {
                __smatch_value("one");
                __smatch_value("two");
        }
        __smatch_value("one");
        __smatch_value("two");
        if (one == 2) {
                __smatch_value("one");
                __smatch_value("two");
        }
        __smatch_value("one");
        __smatch_value("two");
        return 0;
}
/*
 * check-name: smatch equivalent variables #1
 * check-command: smatch -I.. -m64 sm_equiv1.c
 *
 * check-output-start
sm_equiv1.c:13 func() one = 1
sm_equiv1.c:14 func() two = 1
sm_equiv1.c:16 func() one = s64min-s64max
sm_equiv1.c:17 func() two = s64min-s64max
sm_equiv1.c:19 func() one = 2
sm_equiv1.c:20 func() two = 2
sm_equiv1.c:22 func() one = s64min-s64max
sm_equiv1.c:23 func() two = s64min-s64max
 * check-output-end
 */
```

```
**********************************************************
     544 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_equiv2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int *something();

int red;
int blue;
int x;
int func(void)
{
        red = 0;

        if (x) {
                red = 5;
        }
        blue = red;

        if (x) {
                __smatch_value("red");
                __smatch_value("blue");
        }
        __smatch_value("red");
        __smatch_value("blue");
        return 0;
}
/*
 * check-name: smatch equivalent variables #2 (implications)
 * check-command: smatch -I.. sm_equiv2.c
 *
 * check-output-start
sm_equiv2.c:18 func() red = 5
sm_equiv2.c:19 func() blue = 5
sm_equiv2.c:21 func() red = 0,5
sm_equiv2.c:22 func() blue = 0,5
 * check-output-end
 */
```

```
**********************************************************
     511 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_equiv3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
#include "check_debug.h"

int *something();
void frob();

int red;
int blue;
int x;
int func(void)
{

        red = something();
        if (x < 4)
                red = something();
        else if (x > 5)
                red = 0;

        blue = red;
        red = 0;
        if (!blue)
                return;
        __smatch_value("red");
        __smatch_value("blue");
        return 0;
}
/*
 * check-name: smatch equivalent variables #3
 * check-command: smatch -I.. sm_equiv3.c
 *
 * check-output-start
sm_equiv3.c:22 func() red = 0
sm_equiv3.c:23 func() blue = s32min-(-1),1-s32max
 * check-output-end
 */
```

```
**********************************************************
     383 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_equiv4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void *ioremap();
void iounmap(void *);

int *a, *b, *c;
int func(void)
{
        a = ioremap();
        b = ioremap();
        c = a;
        iounmap(c);
        return -1;
}
/*
 * check-name: smatch equivalent variables #4
 * check-command: smatch -p=kernel --spammy -I.. sm_equiv4.c
 *
 * check-output-start
sm_equiv4.c:13 func() warn: 'b' was not released on error
 * check-output-end
 */
```

```
**********************************************************
     368 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_err_ptr.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>

int *add_inode();
int IS_ERR(void *);

int main(void)
{
        int *p;

        p = add_inode();
        if (!IS_ERR(p)) {
                *p = 1;
        }
        *p = 1;

        return 0;
}
/*
 * check-name: dereferencing ERR_PTR bugs
 * check-command: smatch -p=kernel sm_err_ptr.c
 *
 * check-output-start
sm_err_ptr.c:14 main() error: 'p' dereferencing possible ERR_PTR()
 * check-output-end
 */
```

```
**********************************************************
     494 Fri Dec 21 15:01:06 2018
new/usr/src/tools/smatch/src/validation/sm_fake_assignment.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

struct ture {
        int x, y;
};

struct ture outside = {
        .x = 1,
        .y = 2,
};

struct ture buf[10];
void test(void)
{
        int a, b;

        a = 0;
        b = 0;
        buf[a++] = outside;
        buf[++b] = outside;
        __smatch_implied(a);
        __smatch_implied(b);
}

/*
 * check-name: smatch fake assignment
 * check-command: smatch -I.. sm_fake_assignment.c
 *
 * check-output-start
sm_fake_assignment.c:21 test() implied: a = '1'
sm_fake_assignment.c:22 test() implied: b = '1'
 * check-output-end
 */
```

```
**********************************************************
     738 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_get_user1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int frob(void);
#define get_user(x, y) ({ int __val_gu = frob(); x = __val_gu; 0; })

void func(void)
{
        int *user_ptr;
        int foo, bar;
        unsigned int x;

        get_user(foo, user_ptr);
        bar = foo + 1;

        get_user(bar, user_ptr);
        if (bar > foo)
                bar = foo;
        foo = bar * 8;

        get_user(x, user_ptr);
        if (x > foo)
                x = foo;
        foo = x * 8;

        get_user(x, user_ptr);
        foo = x * 8;
}
/*
 * check-name: smatch get_user() #1
 * check-command: smatch -p=kernel -I.. sm_get_user1.c
 *
 * check-output-start
sm_get_user1.c:13 func() warn: check for integer over/underflow 'foo'
sm_get_user1.c:18 func() warn: check for integer underflow 'bar'
sm_get_user1.c:26 func() warn: check for integer overflow 'x'
 * check-output-end
 */
```

```
**********************************************************
     411 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct ture {
   2         int a;
   3 };

   5 struct ture *a;
   6 struct ture *b;

   8 void func (void)
   9 {
  10         struct ture *aa;

  12         b = 0;
  13         if (a)
  14                 goto x;
  15         aa = returns_nonnull();
  16         b = 1;
  17 x:
  18         if (b)
  19                 aa->a = 1;
  20         aa->a = 1;
  21         return;
  22 }
  23 /*
  24  * check-name: Smatch implied #1
  25  * check-command: smatch --spammy sm_implied.c
  26  *
  27  * check-output-start
  28 sm_implied.c:20 func() error: potentially dereferencing uninitialized 'aa'.
  29  * check-output-end
  30  */
```

```
**********************************************************
     690 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied10.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void frob(void){}

int x[10];
int offset;
void func(int *y)
{
        if (({int test2 = !!(!y || !*y); frob(); frob(); frob(); test2;}))
                __smatch_value("y");
        else
                __smatch_value("y");

        if (({int test2 = !!(offset >= 10 || x[offset] == 1); frob(); frob(); fr
                __smatch_value("offset");
        else
                __smatch_value("offset");

}
/*
 * check-name: smatch implied #10
 * check-command: smatch -I.. -m64 sm_implied10.c
 *
 * check-output-start
sm_implied10.c:10 func() y = 0,4096-2117777777777777777
sm_implied10.c:12 func() y = 4096-2117777777777777777
sm_implied10.c:15 func() offset = 0-s32max
sm_implied10.c:17 func() offset = 0-9
 * check-output-end
 */
```

```
**********************************************************
     468 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied11.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

struct foo {
        int x;
};

void *frob();

struct foo *foo;

static void ad_agg_selection_logic(void)
{
        int a;


        if (foo && foo->x)
                a = 1;
        else
                a = 0;

        if (frob())
                a = frob();

        if (a)
                __smatch_implied(foo);
}
/*
 * check-name: smatch implied #11
 * check-command: smatch -I.. -m64 sm_implied11.c
 *
 * check-output-start
sm_implied11.c:25 ad_agg_selection_logic() implied: foo = '0,4096-21177777777777
 * check-output-end
 */
```

**********
497 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied12.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#include "check_debug.h"

struct foo {
        int x;
};

void *frob();

struct foo *foo;

int x;

static void ad_agg_selection_logic(void)
{
        int a;


        if (x) {
                if (foo && foo->x)
                        a = 1;
                else
                        a = 0;

                if (frob())
                        a = frob();

                if (a)
                        __smatch_implied(foo);
        }
}
/*
 * check-name: smatch implied #12
 * check-command: smatch -I.. -m64 sm_implied12.c
 *
 * check-output-start
sm_implied12.c:28 ad_agg_selection_logic() implied: foo = '0,4096-21177777777777
 * check-output-end
 */
```

```
**********************************************************
     381 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied13.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int main(int x)
{
        int a = 1;

        if (x & 12)
                a = 2;
        __smatch_implied(a);
        if (!(x & 12))
                return 0;
        __smatch_implied(a);
        return 0;
}

/*
 * check-name: smatch implied #13
 * check-command: smatch -I.. sm_implied13.c
 *
 * check-output-start
sm_implied13.c:9 main() implied: a = '1-2'
sm_implied13.c:12 main() implied: a = '2'
 * check-output-end
 */
```

```
**********************************************************
     806 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied14.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

static int test(int x)
{
        if (x == 12)
                return 10;
        return 0;
}

int a, b;
static void func(void)
{
        if (a == 12)
                b = 1;
        else
                b = 4;
        if (test(a) == 10) {
                __smatch_implied(a);
                __smatch_implied(b);
        } else {
                __smatch_implied(a);
                __smatch_implied(b);
        }

        if (a == 12)
                b = 10;
        else
                b = 40;

        if (test(a))
                __smatch_implied(b);
        else
                __smatch_implied(b);
}
/*
 * check-name: smatch implied #14
 * check-command: smatch -I.. sm_implied14.c
 *
 * check-output-start
sm_implied14.c:18 func() implied: a = '12'
sm_implied14.c:19 func() implied: b = '1'
sm_implied14.c:21 func() implied: a = 's32min-11,13-s32max'
sm_implied14.c:22 func() implied: b = '4'
sm_implied14.c:31 func() implied: b = '10'
sm_implied14.c:33 func() implied: b = '40'
 * check-output-end
 */
```

```
**********************************************************
     586 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied15.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int some_func(void);

int a;
int frob(int *p)
{
        int ret = 0;

        *p = 4;
        if (a)
                goto out;

        *p = some_func();
        if (*p < 10 || *p > 100) {
                ret = -12;
                goto out;
        }

out:
        return ret;
}

void test(void)
{
        int var = 0;
        int ret;

        ret = frob(&var);
        __smatch_implied(var);
        if (ret)
                return;
        __smatch_implied(var);
}
/*
 * check-name: smatch implied #15
 * check-command: smatch -I.. sm_implied15.c
 *
 * check-output-start
sm_implied15.c:30 test() implied: var = 's32min-s32max'
sm_implied15.c:33 test() implied: var = '4,10-100'
 * check-output-end
 */
```

```
**********************************************************
     630 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied16.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int frob(void);

int a;
int func (char *input)
{
        int x = frob();

        if (a == 1) {
                if (x != -5)
                        return;
        } else if (a == 2) {
                if (x != 0)
                        return;
        } else if (a == 3) {
                if (x != 42)
                        return;
        } else {
                return;
        }

        if (x) {
                __smatch_implied(x);
                __smatch_implied(a);
        }

        if (x == -5)
                __smatch_implied(a);

        return 0;
}

/*
 * check-name: smatch implied #16
 * check-command: smatch -I.. sm_implied16.c
 *
 * check-output-start
sm_implied16.c:24 func() implied: x = '(-5),42'
sm_implied16.c:25 func() implied: a = '1,3'
sm_implied16.c:29 func() implied: a = '1'
 * check-output-end
 */
```

```
**********************************************************
     452 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied17.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int frob(void);

int a;
int func (char *input)
{
        int x = frob();

        if (a == 1) {
                if (x != -5)
                        return;
        } else if (a == 2) {
                if (x < 0 || x > 10)
                        return;
        } else {
                return;
        }

        if (x)
                ;

        if (x == -5)
                __smatch_implied(a);

        return 0;
}

/*
 * check-name: smatch implied #17
 * check-command: smatch -I.. sm_implied17.c
 *
 * check-output-start
sm_implied17.c:24 func() implied: a = '1'
 * check-output-end
 */
```

**********
434 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied18.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#include "check_debug.h"

int a, b;

int frob(int *data)
{
        if (a)
                return 0;
        if (b)
                return -1;
        *data = 42;
        return 1;
}

void test(void)
{
        int x = -1;
        int ret;

        ret = frob(&x);
        if (ret < 0)
                return;
        if (ret == 0)
                return;
        __smatch_implied(x);
}

/*
 * check-name: smatch implied #18
 * check-command: smatch -I.. sm_implied18.c
 *
 * check-output-start
sm_implied18.c:25 test() implied: x = '42'
 * check-output-end
 */
```

```
**********************************************************
     426 Fri Dec 21 15:01:07 2018
new/usr/src/tools/smatch/src/validation/sm_implied19.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int xxx, yyy;
int aaa, bbb;
int id, vbus;
void frob(void)
{
        if (xxx)
                id = yyy;
        else
                id = 1;

        if (aaa)
                vbus = bbb;
        else
                vbus = id;

        if (id)
                ;
        if (!vbus)
                ;

        if (!id)
                __smatch_implied(vbus);
}

/*
 * check-name: smatch implied #19
 * check-command: smatch -I.. sm_implied19.c
 *
 * check-output-start
sm_implied19.c:24 frob() implied: vbus = 's32min-s32max'
 * check-output-end
 */
```

**********
564 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
struct ture {
        int a;
};

struct ture *a;
struct ture *b;
struct ture *c;

void func (void)
{
        struct ture *aa, *ab;

        b = 0;
        if (a) {
                aa = returns_nonnull();
                ab = returns_nonnull();
        } else {
                b = -1;
        }
        if (!(b)) {
                if (c) {
                        aa = (void *)0;
                        ab = (void *)0;
                        b = -1;
                }
        }
        if (!c)
                aa->a = 1;
        if (b)
                return;
        ab->a = 1;
        return;
}
/*
 * check-name: Smatch implied #2
 * check-command: smatch --spammy sm_implied2.c
 *
 * check-output-start
sm_implied2.c:28 func() error: potentially dereferencing uninitialized 'aa'.
 * check-output-end
 */
```

```
**********************************************************
     376 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define NULL ((void *)0)

struct ture {
        int *a;
};

struct ture *b;
struct ture *c;

void func (void)
{
        struct ture *ab;
        int ret = 0;

        if (b) {
                ret = -1;
                goto foo;
        }

        if (c) {}

        ab = some_func();
        if (NULL == ab) {
                ret = -1;
                goto foo;
        }
foo:
        if (ret) {
                return;
        }
        ab->a = 1;
}
/*
 * check-name: Smatch implied #3
 * check-command: smatch sm_implied3.c
 */
```

```
**********************************************************
     610 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void _spin_lock(int name);
   2 void _spin_unlock(int name);

   4 void frob(void){}
   5 int a;
   6 int b;
   7 int c;
   8 int func (void)
   9 {
  10         int mylock = 1;
  11         int mylock2 = 2;

  13         if (a == 3)
  14                 _spin_lock(mylock);
  15         if (b)
  16                 frob();
  17         if (a == 3)
  18                 _spin_unlock(mylock);
  19         if (a)
  20                 _spin_lock(mylock);
  21         if (c)
  22                 return 0;
  23         if (!!a)
  24                 _spin_unlock(mylock);
  25         return 0;
  26 }

  28 /*
  29  * check-name: Smatch implied #4
  30  * check-command: smatch --project=kernel --spammy sm_implied4.c
  31  *
  32  * check-output-start
  33 sm_implied4.c:22 func() warn: 'spin_lock:mylock' is sometimes locked here and so
  34  * check-output-end
  35  */
```

```
**********************************************************
     401 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct ture {
   2         int a;
   3 };

   5 int out_a;

   7 void func (void)
   8 {
   9         struct ture *aa;
  10         int a = 0;

  12         if (out_a) {
  13                 aa = returns_nonnull();
  14                 a = something();
  15         }
  16         if (a)
  17                 aa->a = 1;
  18         aa->a = 0xF00D;
  19 }
  20 /*
  21  * check-name: Smatch implied #5
  22  * check-command: smatch --spammy sm_implied5.c
  23  *
  24  * check-output-start
  25 sm_implied5.c:18 func() error: potentially dereferencing uninitialized 'aa'.
  26  * check-output-end
  27  */
```

```
**********************************************************
     566 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct foo {
   2         int a;
   3 };

   5 struct foo *a;
   6 struct foo *b;
   7 struct foo *c;
   8 struct foo *d;
   9 int x, y, z;

  11 void func (void)
  12 {
  13         a = 0;
  14         b = 0;
  15         c = 0;
  16         d = 0;

  18         if (x)
  19                 a = returns_nonnull();
  20         else
  21                 b = returns_nonnull();
  22         if (y)
  23                 a = returns_nonnull();
  24         else
  25                 c = returns_nonnull();
  26         __smatch_extra_values();
  27         if (x || y) {
  28                 a->a = 1;
  29                 b->a = 2;
  30         }else {
  31                 c->a = 3;
  32         }
  33 }
  34 /*
  35  * check-name: Smatch implied #6
  36  * check-command: smatch --spammy sm_implied6.c
  37  *
  38  * check-output-start
  39 sm_implied6.c:29 func() error: potential NULL dereference 'b'.
  40  * check-output-end
  41  */
```

```
**********************************************************
     417 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied7.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"
int a, b, c;
int frob(void) {
        if (a && b != 1)
                return;

        __smatch_value("a");
        if (b == 0 && c) {
                __smatch_value("a");
        }
        __smatch_value("a");
}
/*
 * check-name: Smatch implied #7
 * check-command: smatch -I.. sm_implied7.c
 *
 * check-output-start
sm_implied7.c:7 frob() a = s32min-s32max
sm_implied7.c:9 frob() a = 0
sm_implied7.c:11 frob() a = s32min-s32max
 * check-output-end
 */
```

```
**********************************************************
     840 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied8.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void frob();

int test, test2;

int x;
void func (void)
{
        if (({int test = !!x; frob(); frob(); frob(); test;}))
                __smatch_value("x");
        else
                __smatch_value("x");
        if (test)
                __smatch_value("x");
        if (({test2 = !(x == 3); frob(); frob(); frob(); test2;}))
                __smatch_value("x");
        else
                __smatch_value("x");
        test = !!(x == 10);
        if (!test)
                __smatch_value("x");
        __smatch_value("x");
}
/*
 * check-name: smatch implied #8
 * check-command: smatch -I.. sm_implied8.c
 *
 * check-output-start
sm_implied8.c:11 func() x = s32min-(-1),1-s32max
sm_implied8.c:13 func() x = 0
sm_implied8.c:15 func() x = s32min-(-1),1-s32max
sm_implied8.c:17 func() x = s32min-2,4-s32max
sm_implied8.c:19 func() x = 3
sm_implied8.c:22 func() x = s32min-9,11-s32max
sm_implied8.c:23 func() x = s32min-s32max
 * check-output-end
 */
```

```
**********************************************************
     715 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_implied9.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void frob(void){}

void func(int y)
{
        int test2;

        if (({int test2 = !!(y < 0 || y >= 10); frob(); frob(); frob(); test2;})
                __smatch_value("y");
        else
                __smatch_value("y");

        test2 = (y < 3 || y >= 5);
        if (test2)
                __smatch_value("y");
        else
                __smatch_value("y");

        if (({int test3 = y < -98; frob(); frob(); frob(); test3;}))
                __smatch_value("y");
}
/*
 * check-name: smatch implied #9
 * check-command: smatch -I.. sm_implied9.c
 *
 * check-output-start
sm_implied9.c:10 func() y = s32min-(-1),10-s32max
sm_implied9.c:12 func() y = 0-9
sm_implied9.c:16 func() y = s32min-2,5-s32max
sm_implied9.c:18 func() y = 3-4
sm_implied9.c:21 func() y = s32min-(-99)
 * check-output-end
 */
```

```
**********************************************************
     505 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_impossible1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int zero(void)
{
        return 0;
}


int main(unsigned int x, unsigned int y)
{
        if (zero())
                __smatch_states("register_impossible_return");
        else
                __smatch_states("register_impossible_return");
}

/*
 * check-name: smatch impossible #1
 * check-command: smatch -I.. sm_impossible1.c
 *
 * check-output-start
sm_impossible1.c:12 main() [register_impossible_return] 'impossible' = 'impossib
sm_impossible1.c:14 main() register_impossible_return: no states
 * check-output-end
 */
```

```
**********************************************************
     503 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_impossible2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int one(void)
{
        return 1;
}


int main(unsigned int x, unsigned int y)
{
        if (one())
                __smatch_states("register_impossible_return");
        else
                __smatch_states("register_impossible_return");
}

/*
 * check-name: smatch impossible #2
 * check-command: smatch -I.. sm_impossible2.c
 *
 * check-output-start
sm_impossible2.c:12 main() register_impossible_return: no states
sm_impossible2.c:14 main() [register_impossible_return] 'impossible' = 'impossib
 * check-output-end
 */
```

```
**********************************************************
     422 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_impossible3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int frob(void);

int main(void)
{
        int x;

        x = frob();

        if (x != -28)
                return;

        if (x != -28 && x != -30)
                __smatch_implied(x);
        __smatch_implied(x);

        return 0;
}

/*
 * check-name: smatch impossible #3
 * check-command: smatch -I.. sm_impossible3.c
 *
 * check-output-start
sm_impossible3.c:15 main() implied: x = ''
sm_impossible3.c:16 main() implied: x = '(-28)'
 * check-output-end
 */
```

```
**********************************************************
     319 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_indirection1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 #include "check_debug.h"

 3 int a;

 5 int frob(int size)
 6 {
 7         int *p = &a;

 9         *p = 42;
10         __smatch_implied(a);

12         return 0;
13 }

15 /*
16  * check-name: smatch: pointer indirection #1
17  * check-command: smatch -p=kernel -I.. sm_indirection1.c
18  *
19  * check-output-start
20 sm_indirection1.c:10 frob() implied: a = '42'
21  * check-output-end
22  */
```

```
**********************************************************
     618 Fri Dec 21 15:01:08 2018
new/usr/src/tools/smatch/src/validation/sm_indirection2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

struct foo {
        int a, b, c;
};

struct bar {
        struct foo *foo;
};

struct foo *get_foo(struct bar *bar)
{
        return bar->foo;
}

void frob(struct bar *bar)
{
        struct foo *f = bar->foo;
        f->a = 5;
}

int test(struct bar *bar)
{
        struct foo *f = get_foo(bar);

        f->a = 1;
        frob(bar);
        __smatch_implied(bar->foo->a);
        __smatch_implied(f->a);

        return 0;
}

/*
 * check-name: smatch: indirection #2
 * check-command: smatch -I.. sm_indirection2.c
 *
 * check-output-start
sm_indirection2.c:28 test() implied: bar->foo->a = '5'
sm_indirection2.c:29 test() implied: f->a = '5'
 * check-output-end
 */
```

```
**********************************************************
     658 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_initializer.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

struct bar {
        int a, b, c;
};

struct foo {
        struct bar bar;
        int x, y, z;
};

int test(int size)
{
        struct foo foo = {
                .bar.a = 42,
                .bar.b = 43,
                -1,
        };
        __smatch_implied(foo.bar.b);
        __smatch_implied(foo.bar.c);
        __smatch_implied(foo.x);
        __smatch_implied(foo.y);

        return 0;
}

/*
 * check-name: smatch: nested initializer
 * check-command: smatch -I.. sm_initializer.c
 *
 * check-output-start
sm_initializer.c:19 test() implied: foo.bar.b = '43'
sm_initializer.c:20 test() implied: foo.bar.c = '0'
sm_initializer.c:21 test() implied: foo.x = '(-1)'
sm_initializer.c:22 test() implied: foo.y = '0'
 * check-output-end
 */
```

```
**********************************************************
     601 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_inline1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

int frob(int *x)
{
        *x = *x * 3;
        return 0;
}

int *x;
int y;
int main(void)
{
        *x = 1;
        frob(x);
        __smatch_implied(*x);
        frob(x);
        __smatch_implied(*x);

        y = 2;
        frob(&y);
        __smatch_implied(y);
        frob(&y);
        __smatch_implied(y);

        return 0;
}


/*
 * check-name: smatch: inline #1
 * check-command: smatch -I.. sm_inline1.c
 *
 * check-output-start
sm_inline1.c:17 main() implied: *x = '3'
sm_inline1.c:19 main() implied: *x = '9'
sm_inline1.c:23 main() implied: y = '6'
sm_inline1.c:25 main() implied: y = '18'
 * check-output-end
 */
```

```
**********************************************************
     392 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_inline2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include <string.h>
#include "check_debug.h"

int frob(int *x)
{
        *x = *x * 3;
        return 0;
}

int *x;
int main(void)
{
        frob(x);
        if (x)
                return 1;
        return 0;
}


/*
 * check-name: smatch: inline #2
 * check-command: smatch -I.. sm_inline2.c
 *
 * check-output-start
sm_inline2.c:15 main() warn: variable dereferenced before check 'x' (see line 14
 * check-output-end
 */
```

```
**********************************************************
     710 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_inline3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void memset(void *p, char pat, int size);

struct foo {
        int a, b;
};

void my_func(struct foo *p)
{
        memset(p, 0, sizeof(*p));
        p->a = 1;
}

struct foo *my_pointer;

void test(void)
{
        struct foo foo;

        my_func(my_pointer);
        my_func(&foo);
        __smatch_implied(my_pointer->a);
        __smatch_implied(my_pointer->b);
        __smatch_implied(foo.a);
        __smatch_implied(foo.b);
}

/*
 * check-name: smatch: inline #3
 * check-command: smatch -I.. sm_inline3.c
 *
 * check-output-start
sm_inline3.c:23 test() implied: my_pointer->a = '1'
sm_inline3.c:24 test() implied: my_pointer->b = '0'
sm_inline3.c:25 test() implied: foo.a = '1'
sm_inline3.c:26 test() implied: foo.b = '0'
 * check-output-end
 */
```

```
**********************************************************
     690 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_locking.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 void _spin_lock(int name);
 2 void _spin_unlock(int name);

 4 int a, b, c;

 6 int func (void)
 7 {
 8         int mylock = 1;
 9         int mylock2 = 1;
10         int mylock3 = 1;

12         if (a) {
13                 return;
14         }

16         _spin_lock(mylock);
17         _spin_unlock(mylock);

19         if (b) {
20                 _spin_unlock(mylock2);
21                 return;
22         }

24         if (c)
25                 _spin_lock(mylock3);
26         return;
27 }
28 /*
29  * check-name: Smatch locking #1
30  * check-command: smatch --project=kernel --spammy sm_locking.c
31  *
32  * check-output-start
33 sm_locking.c:26 func() warn: 'spin_lock:mylock3' is sometimes locked here and so
34 sm_locking.c:26 func() warn: inconsistent returns 'spin_lock:mylock2'.
35   Locked on:   line 13
36                line 26
37   Unlocked on: line 21
38  * check-output-end
39  */
```

```
**********************************************************
     717 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_locking2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void _spin_lock(int name);
   2 void _spin_unlock(int name);
   3 int _spin_trylock(int name);

   5 int a;
   6 int b;
   7 int func (void)
   8 {
   9         int mylock = 1;
  10         int mylock2 = 1;
  11         int mylock3 = 1;

  13         if (!_spin_trylock(mylock)) {
  14                 return;
  15         }

  17         _spin_unlock(mylock);
  18         _spin_unlock(mylock2);

  20         if (a)
  21                 _spin_unlock(mylock);
  22         _spin_lock(mylock2);

  24         if (!_spin_trylock(mylock3))
  25                 return;
  26         return;
  27 }
  28 /*
  29  * check-name: Smatch locking #2
  30  * check-command: smatch --project=kernel sm_locking2.c
  31  *
  32  * check-output-start
  33 sm_locking2.c:21 func() error: double unlock 'spin_lock:mylock'
  34 sm_locking2.c:26 func() warn: inconsistent returns 'spin_lock:mylock3'.
  35   Locked on:   line 26
  36   Unlocked on: line 14
  37                line 25
  38  * check-output-end
  39  */
```

```
**********************************************************
     665 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_locking3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int _spin_trylock(int name);
void _spin_lock(int name);
void _spin_unlock(int name);

int func (void)
{
        int mylock = 1;

        if (!({frob(); frob(); _spin_trylock(mylock);}))
                return;

        frob();
        _spin_unlock(mylock);

        if (((_spin_trylock(mylock)?1:0)?1:0))
                return;
        frob_somemore();
        _spin_unlock(mylock);

        return;
}
/*
 * check-name: Smatch locking #3
 * check-command: smatch --project=kernel sm_locking3.c
 *
 * check-output-start
sm_locking3.c:18 func() error: double unlock 'spin_lock:mylock'
sm_locking3.c:20 func() warn: inconsistent returns 'spin_lock:mylock'.
  Locked on:   line 16
  Unlocked on: line 10
               line 20
 * check-output-end
 */
```

```
**********************************************************
     622 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_locking4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void _spin_lock(int name);
   2 void _spin_unlock(int name);

   4 void frob(void){}
   5 int a;
   6 int b;
   7 void func (void)
   8 {
   9         int mylock = 1;
  10         int mylock2 = 2;

  12         if (1)
  13                 _spin_unlock(mylock);
  14         frob();
  15         if (a)
  16                 return;
  17         if (!0)
  18                 _spin_lock(mylock);
  19         if (0)
  20                 _spin_unlock(mylock);
  21         if (b)
  22                 return;
  23         if (!1)
  24                 _spin_lock(mylock);
  25 }
  26 /*
  27  * check-name: Smatch locking #4
  28  * check-command: smatch --project=kernel sm_locking4.c
  29  *
  30  * check-output-start
  31 sm_locking4.c:23 func() warn: inconsistent returns 'spin_lock:mylock'.
  32   Locked on:   line 22
  33                line 23
  34   Unlocked on: line 16
  35  * check-output-end
  36  */
```

```
**********************************************************
    1002 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_locking6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int __raw_local_irq_save();
int _spin_trylock(int x);
int raw_local_irq_restore(flags);

#define spin_trylock_irqsave(lock, flags) \
({ \
        (flags) = __raw_local_irq_save(); \
        _spin_trylock(lock) ? 1 : ({ raw_local_irq_restore(flags);  0; }); \
})

void _spin_unlock_irqrestore(int lock, int flags);

int zzz;

void func (void)
{
        int lock = 1;
        int flags = 1;

        if (!spin_trylock_irqsave(lock, flags))
                return;
        _spin_unlock_irqrestore(lock, flags);
        if (zzz)
                return;
        if (spin_trylock_irqsave(lock, flags))
                return;
        return;
}
/*
 * check-name: Smatch locking #6
 * check-command: smatch -p=kernel sm_locking6.c
 *
 * check-output-start
sm_locking6.c:27 func() warn: inconsistent returns 'irqsave:flags'.
  Locked on:   line 26
  Unlocked on: line 21
               line 24
               line 27
sm_locking6.c:27 func() warn: inconsistent returns 'spin_lock:lock'.
  Locked on:   line 26
  Unlocked on: line 21
               line 24
               line 27
 * check-output-end
 */
```

```
**********************************************************
     743 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_locking7.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

unsigned long arch_local_irq_save();
void arch_local_irq_restore(unsigned long flags);
int spin_trylock();
void frob();

void func(int *y)
{
        int lock;
        unsigned long flags;

        if (({
                int __ret;
                flags = arch_local_irq_save();
                __ret = spin_trylock(&lock);
                if (!__ret)
                        arch_local_irq_restore(flags);
                __ret;
            }))
                return;
        frob();
}

/*
 * check-name: smatch locking #7
 * check-command: smatch -p=kernel -I.. sm_locking7.c
 *
 * check-output-start
sm_locking7.c:22 func() warn: inconsistent returns 'irqsave:flags'.
  Locked on:   line 21
  Unlocked on: line 22
sm_locking7.c:22 func() warn: inconsistent returns 'spin_lock:&lock'.
  Locked on:   line 21
  Unlocked on: line 22
 * check-output-end
 */
```

```
**********************************************************
     730 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_loops1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void frob();

void func(int *x)
{
        int a,b,c;

        for (a = 0; a < 10; a++)
                __smatch_value("a");
        __smatch_value("a");
        for (; a != 10; a++)
                __smatch_value("a");
        __smatch_value("a");
        for (a = 0; a != 10; a++)
                __smatch_value("a");
        __smatch_value("a");
        for (a = 0; a <= 10; a++)
                __smatch_value("a");
        __smatch_value("a");
        return;
}
/*
 * check-name: smatch loops #1
 * check-command: smatch -I.. sm_loops1.c
 *
 * check-output-start
sm_loops1.c:10 func() a = 0-9
sm_loops1.c:11 func() a = 10
sm_loops1.c:13 func() a = empty
sm_loops1.c:14 func() a = 10
sm_loops1.c:16 func() a = 0-9
sm_loops1.c:17 func() a = 10
sm_loops1.c:19 func() a = 0-10
sm_loops1.c:20 func() a = 11
 * check-output-end
 */
```

```
**********************************************************
     834 Fri Dec 21 15:01:09 2018
new/usr/src/tools/smatch/src/validation/sm_loops2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int checker(void);

int x;
int y;
void func(void)
{
        while (x--)
                __smatch_value("x");
        __smatch_value("x");
        for (x = 0; x < y; x++) {
                if (checker())
                        break;
        }
        __smatch_value("x");
        while (x--)
                __smatch_value("x");
        __smatch_value("x");
        x = 10;
        while (x--)
                __smatch_value("x");
        __smatch_value("x");
        x = 10;
        while (--x)
                __smatch_value("x");
        __smatch_value("x");
}
/*
 * check-name: smatch loops #1
 * check-command: smatch -I.. sm_loops2.c
 *
 * check-output-start
sm_loops2.c:10 func() x = s32min-s32max
sm_loops2.c:11 func() x = s32min-s32max
sm_loops2.c:16 func() x = 0-s32max
sm_loops2.c:18 func() x = 0-s32max
sm_loops2.c:19 func() x = (-1)
sm_loops2.c:22 func() x = 0-9
sm_loops2.c:23 func() x = (-1)
sm_loops2.c:26 func() x = 1-9
sm_loops2.c:27 func() x = 0
 * check-output-end
 */
```

```
**********************************************************
     360 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_loops3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int checker(void);

int x;
int i;
void func(void)
{
        int ar[10];

        if (i < 0)
                return;
        if(i == 0)
                x = 11;
        else
                x = 1;

        while(i--) {
                __smatch_value("x");
                ar[x] = 1;
        }
}
/*
 * check-name: smatch loops #3
 * check-command: smatch -I.. sm_loops3.c
 *
 * check-output-start
sm_loops3.c:19 func() x = 1
 * check-output-end
 */
```

```
**********************************************************
     451 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_loops4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

char *some_func(void);

int x,y;
int i;
void func(void)
{
        char *p;
        char *p2;

        if (x > 0)
                p = some_func();
        for (i = 0; i < x; i++)
                *p = 'x';
        *p = 'x';
        if (y > 0)
                p2 = some_func();
        i = 0;
        if (i < y)
                *p2 = 'x';
}
/*
 * check-name: smatch loops #4
 * check-command: smatch -I.. sm_loops4.c
 *
 * check-output-start
sm_loops4.c:16 func() error: potentially dereferencing uninitialized 'p'.
 * check-output-end
 */
```

```
**********************************************************
     306 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_loops5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int frob(void);

int a, b, c;
void test(void)
{
        a = 0;
        do {
                frob();
        } while (a++ < 3);
        __smatch_implied(a);
}
/*
 * check-name: smatch loops #5
 * check-command: smatch -I.. sm_loops5.c
 *
 * check-output-start
sm_loops5.c:12 test() implied: a = '4'
 * check-output-end
 */
```

```
**********************************************************
    1013 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_loops6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

extern int xxx;

int test(struct bar *bar)
{
        int a = 10, b = 10, c = 10, d = 10, e = 10;

        while (a--)
                __smatch_implied(a);
        __smatch_implied(a);

        while (--b)
                __smatch_implied(b);
        __smatch_implied(b);

        while (--c >= 0)
                __smatch_implied(c);
        __smatch_implied(c);

        while (d-- >= 0)
                __smatch_implied(d);
        __smatch_implied(d);

        while (e-- >= 0) {
                if (xxx)
                        break;
                __smatch_implied(e);
        }
        __smatch_implied(e);

        return 0;
}


/*
 * check-name: smatch loops #6
 * check-command: smatch -I.. sm_loops6.c
 *
 * check-output-start
sm_loops6.c:10 test() implied: a = '0-9'
sm_loops6.c:11 test() implied: a = '(-1)'
sm_loops6.c:14 test() implied: b = '1-9'
sm_loops6.c:15 test() implied: b = '0'
sm_loops6.c:18 test() implied: c = '0-9'
sm_loops6.c:19 test() implied: c = '(-1)'
sm_loops6.c:22 test() implied: d = '(-1)-9'
sm_loops6.c:23 test() implied: d = '(-2)'
sm_loops6.c:28 test() implied: e = '(-1)-9'
sm_loops6.c:30 test() implied: e = '(-2)-9'
 * check-output-end
 */
```

```
**********************************************************
     778 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_macros.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

#define add(x, y) x + y
#define sub(x, y) x - y

int x;
void func(int *p)
{
        int a = 1;
        int b = 2;

        x = 4 * add(2, 3);
        x = 4 + add(2, 3);
        x = 4 * add(2, 3) * 8;
        x = add(2, 3) * 4;
        x = add(2, 3) - 4;
        x = -sub(2, 3);
        x = sub(2, 3)++;
}
/*
 * check-name: Smatch macro precedence bugs
 * check-command: smatch -I.. sm_macros.c
 *
 * check-output-start
sm_macros.c:12 func() warn: the 'add' macro might need parens
sm_macros.c:14 func() warn: the 'add' macro might need parens
sm_macros.c:14 func() warn: the 'add' macro might need parens
sm_macros.c:15 func() warn: the 'add' macro might need parens
sm_macros.c:17 func() warn: the 'sub' macro might need parens
sm_macros.c:18 func() warn: the 'sub' macro might need parens
 * check-output-end
 */
```

```
**********************************************************
     611 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_math1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int something();

void func(void)
{
        int x = 3;
        int y = 42;
        int z = 7;

        x--;
        y -= 100;
        __smatch_value("y");
        while (something()) {
                y++;
                __smatch_value("y");
        }
        z += something();
        __smatch_value("z");
        __smatch_value("x");
label_I1:
        x--;
        __smatch_value("x");
        goto label_I1;
}
/*
 * check-name: smatch math test #1
 * check-command: smatch -I.. sm_math1.c
 *
 * check-output-start
sm_math1.c:13 func() y = (-58)
sm_math1.c:16 func() y = (-57)-s32max
sm_math1.c:19 func() z = s32min-s32max
sm_math1.c:20 func() x = 2
sm_math1.c:23 func() x = s32min-1
 * check-output-end
 */
```

```
**********************************************************
     467 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_math2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

unsigned char buf[2];

void test(void)
{
        int a = buf[1];
        int b = buf[0] << 8;
        int c = (buf[0] << 8) | buf[1];

        __smatch_implied(a);
        __smatch_implied(b);
        __smatch_implied(c);
}

/*
 * check-name: smatch math #2
 * check-command: smatch -I.. sm_math2.c
 *
 * check-output-start
sm_math2.c:11 test() implied: a = '0-255'
sm_math2.c:12 test() implied: b = '0,256-65280'
sm_math2.c:13 test() implied: c = '0-u16max'
 * check-output-end
 */
```

```
**********************************************************
     276 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_memleak2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdlib.h>

void func (void)
{
        void *ptr;

        ptr = malloc(42);
        ptr = (void *) 0;

        return;
}
/*
 * check-name: leak test #2
 * check-command: smatch sm_memleak2.c
 *
 * check-output-start
sm_memleak2.c:8 func() warn: overwrite may leak 'ptr'
 * check-output-end
 */
```

```
**********************************************************
     515 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_memory.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 void *kmalloc(void);
  2 void free(void *);

  4 struct ture {
  5         int *a;
  6 };

  8 struct ture *a;
  9 struct ture *b;
 10 void func (void)
 11 {
 12         struct ture *aa;
 13         struct ture *ab;
 14         struct ture *ac;
 15         aa = kmalloc();
 16         ab = kmalloc();
 17         ac = kmalloc();

 19         a = aa;
 20         if (ab) {
 21                 free(ab);
 22                 return;
 23         }
 24         free(ac);
 25         return;
 26 }
 27 /*
 28  * check-name: leak test #1
 29  * check-command: smatch sm_memory.c
 30  *
 31  * check-output-start
 32 sm_memory.c:22 func() warn: possible memory leak of 'ac'
 33 sm_memory.c:22 func() error: memory leak of 'ac'
 34  * check-output-end
 35  */
```

```
**********************************************************
     572 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_mod.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b;

int frob(int size)
{
        if (a <= 0 || a > 10)
                return;
        if (a % 4) {
                __smatch_implied(a);
        } else {
                __smatch_implied(a);
        }

        if (b <= 0 || b > 100)
                return;
        if (b % 4) {
                __smatch_implied(b);
        } else {
                __smatch_implied(b);
        }



        return 0;
}

/*
 * check-name: smatch mod condition
 * check-command: smatch -I.. sm_mod.c
 *
 * check-output-start
sm_mod.c:10 frob() implied: a = '1-10'
sm_mod.c:12 frob() implied: a = '4,8'
sm_mod.c:18 frob() implied: b = '1-99'
sm_mod.c:20 frob() implied: b = '4-100'
 * check-output-end
 */
```

```
**********************************************************
     803 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_mtag1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "check_debug.h"

static int xxx = 234;

struct foo {
        int a, b, c;
        int (*func)(struct foo *p);
};

static int frob1(struct foo *p)
{
        printf("%d\n", p->a);
        __smatch_implied(p->a);
        return p->a + 1;
}
static int frob2(struct foo *p)
{
        printf("%d\n", p->a);
        __smatch_implied(p->a);
        return p->a + 1;
}

static struct foo one_struct = {
        .a = 1,
        .func = frob1,
};
static struct foo two_struct = {
        .a = 2,
        .func = frob2,
};

int main(void)
{
        struct foo *p = &one_struct;
        int ret;

        ret = p->func(p);
//      __smatch_implied(ret);

        return 0;
}

/*
 * check-name: smatch mtag #1
 * check-command: validation/smatch_db_test.sh -I.. sm_mtag1.c
 *
 * check-output-start
sm_mtag1.c:14 frob1() implied: p->a = '1'
sm_mtag1.c:20 frob2() implied: p->a = '2'
 * check-output-end
 */
```

```
**********************************************************
     762 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_mtag2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "check_debug.h"

struct foo {
        int a, b, c;
        int (*func)(int *p);
        void (*func2)(int a);
        void *data;
};

static int frob1(int *val)
{
        __smatch_implied(*val);
        return *val + 1;
}

static int frob2(int *val)
{
        __smatch_implied(*val);
        return *val + 1;
}

static struct foo one_struct = {
        .a = 1,
        .func = frob1,
};

static struct foo two_struct = {
        .a = 2,
        .func = frob2,
};

struct foo *unknown(void);
struct foo *p;

int main(void)
{
        int ret;

        p = unknown();
        ret = p->func(&p->a);

        return 0;
}

/*
 * check-name: smatch mtag #2
 * check-command: validation/smatch_db_test.sh -I.. sm_mtag2.c
 *
 * check-output-start
sm_mtag2.c:13 frob1() implied: *val = '1'
sm_mtag2.c:19 frob2() implied: *val = '2'
 * check-output-end
 */
```

```
**********************************************************
     666 Fri Dec 21 15:01:10 2018
new/usr/src/tools/smatch/src/validation/sm_mtag3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int x;
int array[10];

int main(void)
{
        __smatch_implied(&x);
        __smatch_implied((unsigned long)(array + 1) - (unsigned long)array);
        __smatch_implied(array + 1 - array);
        __smatch_implied(array + 1);
        __smatch_implied((int *)0 + 1);

        return 0;
}


/*
 * check-name: smatch mtag #3
 * check-command: smatch -I.. sm_mtag3.c
 *
 * check-output-start
sm_mtag3.c:8 main() implied: &x = '799717014380380160'
sm_mtag3.c:9 main() implied: (array[1]) - array = '4'
sm_mtag3.c:10 main() implied: array[1] - array = '1'
sm_mtag3.c:11 main() implied: array[1] = '7934625272050024452'
sm_mtag3.c:12 main() implied: 0 + 1 = '4'
 * check-output-end
 */
```

```
**********************************************************
     746 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_mtag4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "check_debug.h"

struct foo {
        int a, b, c;
        int (*func)(int *p);
        void (*func2)(int a);
        void *data;
};

static void frob_int1(int val)
{
        __smatch_implied(val);
}

static void frob_int2(int val)
{
        __smatch_implied(val);
}

static struct foo one_struct = {
        .b = 41,
        .func2 = frob_int1,
};

static struct foo two_struct = {
        .b = 42,
        .func2 = frob_int2,
};

struct foo *unknown(void);
struct foo *p;

int main(void)
{
        int ret;

        p = unknown();
        p->func2(p->b);

        return 0;
}

/*
 * check-name: smatch mtag #4
 * check-command: validation/smatch_db_test.sh -I.. sm_mtag4.c
 *
 * check-output-start
sm_mtag4.c:13 frob_int1() implied: val = '41'
sm_mtag4.c:18 frob_int2() implied: val = '42'
 * check-output-end
 */
```

```
**********************************************************
     736 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_mtag5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "check_debug.h"

int x = 42;

struct foo {
        int a, b, c;
};
struct foo aaa = {
        .a = 1, .b = 2, .c = 3,
};

int array[10];

int main(void)
{
        __smatch_implied(&x);
        __smatch_implied(&aaa);
        __smatch_implied(&aaa.b);
        __smatch_implied(array);
        __smatch_implied(&array[1]);

        return 0;
}

/*
 * check-name: smatch mtag #5
 * check-command: smatch -I.. sm_mtag5.c
 *
 * check-output-start
sm_mtag5.c:17 main() implied: &x = '799717014380380160'
sm_mtag5.c:18 main() implied: &aaa = '126458562737565696'
sm_mtag5.c:19 main() implied: &aaa.b = '126458562737565700'
sm_mtag5.c:20 main() implied: array = '7934625272050024448'
sm_mtag5.c:21 main() implied: &array[1] = '7934625272050024452'
 * check-output-end
 */
```

```
**********************************************************
     529 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_mtag6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "check_debug.h"

int frob1(int *p)
{
        __smatch_implied(*p);
}

int frob2(int *p)
{
        __smatch_implied(*p);
}

int x = 42;

struct foo {
        int a, b, c;
};
struct foo aaa = {
        .a = 1, .b = 2, .c = 3,
};

int array[10];

int main(void)
{
        frob1(&x);
        frob2(&aaa.b);

        return 0;
}

/*
 * check-name: smatch mtag #6
 * check-command: validation/smatch_db_test.sh -I.. sm_mtag6.c
 *
 * check-output-start
sm_mtag6.c:6 frob1() implied: *p = '42'
sm_mtag6.c:11 frob2() implied: *p = '2'
 * check-output-end
 */
```

```
**********************************************************
     698 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_mtag7.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "check_debug.h"

struct foo {
        int (*func)(struct foo *);
        int a, b, c;
        int *p;
};

int frob1(struct foo *p)
{
        __smatch_implied(*p->p);
}

int frob2(struct foo *p)
{
        __smatch_implied(*p->p);
}

int x = 42;
int y = 43;

struct foo aaa = {
        .func = frob1,
        .a = 1, .b = 2, .c = 3,
        .p = &x,
};
struct foo bbb = {
        .func = frob2,
        .a = 10, .b = 11, .c = 13,
        .p = &y,
};

int main(void)
{
        aaa.func(&aaa);
        bbb.func(&bbb);
        return 0;
}

/*
 * check-name: smatch mtag #7
 * check-command: validation/smatch_db_test.sh -I.. sm_mtag7.c
 *
 * check-output-start
sm_mtag7.c:12 frob1() implied: *p->p = '42'
sm_mtag7.c:17 frob2() implied: *p->p = '43'
 * check-output-end
 */
```

*********************************************************
*** 648 Fri Dec 21 15:01:11 2018
*new/usr/src/tools/smatch/src/validation/sm_netdevice.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
*********************************************************

```
 2 void kfree(void *);

 4 struct net_device {
 5         char *name;
 6 };

 8 void func(void)
 9 {
10         struct net_device dev;
11         struct net_device *dev2 = &dev;
12         struct net_device **dev3 = &dev2;
13         struct net_device *deva[10];
14         struct net_device **devb[10];
15         struct net_device ***devc = devb;

17         kfree(dev2);
18         kfree(dev3);
19         kfree(deva[0]);
20         kfree(devb[0]);
21         kfree(devc[0]);
22 }
23 /*
24  * check-name: free_netdev() vs kfree()
25  * check-command: smatch -p=kernel sm_netdevice.c
26  *
27  * check-output-start
28 sm_netdevice.c:17 func() error: use free_netdev() here instead of kfree(dev2)
29 sm_netdevice.c:19 func() error: use free_netdev() here instead of kfree(deva[0])
30  * check-output-end
31  */
```

```
**********************************************************
     825 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_null_deref.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

struct foo {
        int a;
};

struct foo *a;
struct foo *b;
struct foo *c;
struct foo *d;

static void func (void)
{
        struct foo *aa;
        int ab = 0;
        int ac = 1;

        aa->a = 1;

        if (a) {
                a->a = 1;
        }
        a->a = 1;

        if (a && b) {
                b->a = 1;
        }

        if (a || b) {
                b->a = 1;
        }

        if (c) {
                ab = 1;
        }

        if (ab) {
                c->a = 1;
        }
}
/*
 * check-name: Null Dereferences
 * check-command: smatch --spammy -I.. sm_null_deref.c
 *
 * check-output-start
sm_null_deref.c:18 func() error: potentially dereferencing uninitialized 'aa'.
sm_null_deref.c:23 func() error: we previously assumed 'a' could be null (see li
sm_null_deref.c:25 func() warn: variable dereferenced before check 'a' (see line
sm_null_deref.c:30 func() error: we previously assumed 'b' could be null (see li
 * check-output-end
 */
```

```
**********************************************************
     546 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_null_deref2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int *ptr;
int x, y, z;
void frob(void) {
        if ((y && !ptr) || z)
                return;
        if (ptr) {
                /* in the current version of smatch this erases
                   the old implications.  Later version should
                   fix this.  --Dec 2 2009 */
                x = *ptr;
        }
        if (!y && ptr)
                *ptr = 0; // smatch used to print an error here.
        if (!y)
                *ptr = 1;
}
/*
 * check-name: Dereferencing Undefined
 * check-command: smatch sm_null_deref2.c
 *
 * check-output-start
sm_null_deref2.c:15 frob() error: we previously assumed 'ptr' could be null (see
 * check-output-end
 */
```

```
**********************************************************
    2582 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_overflow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 struct field {
 2     int b[8];
 3 };
 4
 5 struct buffer {
 6     struct field a;
 7     int x;
 8 };
 9
10 int main(int argc, char* argv[])
11 {
12     struct buffer b1;
13     int i;
14
15     b1.a.b[10] = 1;
16
17     return 42;
18 }
19
20 /*
21  * check-name: Check array overflow
22  * check-command: smatch sm_overflow.c
23  *
24  * check-output-start
25 sm_overflow.c:15 main() error: buffer overflow 'b1.a.b' 8 <= 10
26  * check-output-end
27  */
```

```
**********************************************************
     566 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_overflow3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void strcpy(char *to, char *from, int size);

void func (char *a, char *b)
{
        char c[4];
        char d[4];
        char e[4];
        char f[4];

        b = "1234";
        strcpy(a, b);
        a[5] = '\0';
        strcpy(c, b);
        strcpy(d, "123");
        strcpy(e, "1234");
        strcpy(f, "12");
        f[3] = '\0';
}
/*
 * check-name: smatch strcpy overflow
 * check-command: smatch -I.. sm_overflow3.c
 *
 * check-output-start
sm_overflow3.c:15 func() error: strcpy() 'b' too large for 'c' (5 vs 4)
sm_overflow3.c:17 func() error: strcpy() '"1234"' too large for 'e' (5 vs 4)
 * check-output-end
 */
```

```
**********************************************************
     386 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_overflow4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void strndup(char *to, int size);
void strcpy(char *dest, char *src);

void func (char *a, char *b)
{
        char c[5];

        a = strndup(b, 5);
        strcpy(c, a);
}
/*
 * check-name: smatch strndup overflow
 * check-command: smatch -I.. sm_overflow4.c
 *
 * check-output-start
sm_overflow4.c:11 func() error: strcpy() 'a' too large for 'c' (6 vs 5)
 * check-output-end
 */
```

```
**********************************************************
     384 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_overflow5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void memdup(char *to, int size);
void strcpy(char *dest, char *src);

void func (char *a, char *b)
{
        char c[5];

        a = memdup(b, 5);
        strcpy(c, a);
        a[5] = '\0';
}
/*
 * check-name: smatch memdup overflow
 * check-command: smatch -I.. sm_overflow5.c
 *
 * check-output-start
sm_overflow5.c:12 func() error: buffer overflow 'a' 5 <= 5
 * check-output-end
 */
```

```
**********************************************************
     727 Fri Dec 21 15:01:11 2018
new/usr/src/tools/smatch/src/validation/sm_overflow6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int strlen(char *buf);
void strcpy(char *dest, char *src);
int snprintf(char *dest, int limit, char *format, char *str);
int sprintf(char *dest, char *format, char *str);

char *str;

int main(void)
{
        char buf[10];
        char buf1[10];

        if (strlen(str) > 11)
                return;
        snprintf(buf, 11, "%s", str);
        sprintf(buf1, "%s", str);
}
/*
 * check-name: smatch overflow #6
 * check-command: smatch -I.. sm_overflow6.c
 *
 * check-output-start
sm_overflow6.c:17 main() error: snprintf() is printing too much 11 vs 10
sm_overflow6.c:17 main() error: snprintf() chops off the last chars of 'str': 12
sm_overflow6.c:18 main() error: sprintf() copies too much data from 'str': 12 vs
 * check-output-end
 */
```

```
**********************************************************
     364 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_pointer_assign.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include <stdio.h>
#include "check_debug.h"

int *aaa, *bbb;

int main(void)
{
        if (*aaa < 0 || *aaa > 34)
                return -1;
        bbb = aaa;
        __smatch_implied(*bbb);

        return 0;
}

/*
 * check-name: smatch pointer assign
 * check-command: smatch -I.. sm_pointer_assign.c
 *
 * check-output-start
sm_pointer_assign.c:11 main() implied: *bbb = '0-34'
 * check-output-end
 */
```

```
**********************************************************
     913 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_precedence.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct foo {
        unsigned int x:1;
};

int frob();
int a,b,c,d, x, y;
struct foo *z;
static int options_write(void)
{
        if (x & y == 0)
                frob();
        if (x | y == 0)
                frob();
        if (a == b & c == d)
                frob();
        if (a == c == d)
                frob();
        if (!a == b)
                frob();
        c = !a & b;
        if (x + y == 0)
                frob();
        if (!a == !b)
                frob();
        if (!a == z->x)
                frob();
        if (!!a == b)
                frob();

}
/*
 * check-name: Smatch precedence check
 * check-command: smatch sm_precedence.c
 *
 * check-output-start
sm_precedence.c:10 options_write() warn: add some parenthesis here?
sm_precedence.c:12 options_write() warn: add some parenthesis here?
sm_precedence.c:14 options_write() warn: add some parenthesis here?
sm_precedence.c:16 options_write() warn: add some parenthesis here?
sm_precedence.c:18 options_write() warn: add some parenthesis here?
sm_precedence.c:20 options_write() warn: add some parenthesis here?
 * check-output-end
 */
```

```
**********************************************************
     353 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_range1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 struct ture {
 2         int x;
 3 };

 5 struct ture *p;
 6 struct ture *q;
 7 int xxx;

 9 int func (void)
10 {

12         for (xxx = 0; xxx < 10; xxx++) {
13                 if (p && q)
14                         break;
15         }
16 // this needs two pass processing to work.
17 //      if (xxx == 5)
18 //              q->x = 1;
19         if (xxx == 10)
20                 return;
21         p->x = 1;

23         return 0;
24 }

26 /*
27  * check-name: Implied Ranges #1
28  * check-command: smatch sm_range1.c
29  */
```

```
**********************************************************
     868 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_range2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"
int some_func();
int a, b, c, d, e;
int frob(void) {
        if (a)
                __smatch_value("a");
        else
                __smatch_value("a");
        __smatch_value("a");
        if (a) {
                b = 0;
                __smatch_value("b");
        }
        __smatch_value("b");
        c = 0;
        c = some_func();
        __smatch_value("c");
        if (d < -3 || d > 99)
                return;
        __smatch_value("d");
        if (d) {
                if (!e)
                        return;
        }
        __smatch_value("d");
        __smatch_value("e");
}
/*
 * check-name: Smatch range test #2
 * check-command: smatch -I.. sm_range2.c
 *
 * check-output-start
sm_range2.c:6 frob() a = s32min-(-1),1-s32max
sm_range2.c:8 frob() a = 0
sm_range2.c:9 frob() a = s32min-s32max
sm_range2.c:12 frob() b = 0
sm_range2.c:14 frob() b = s32min-s32max
sm_range2.c:17 frob() c = s32min-s32max
sm_range2.c:20 frob() d = (-3)-99
sm_range2.c:25 frob() d = (-3)-99
sm_range2.c:26 frob() e = s32min-s32max
 * check-output-end
 */
```

```
**********************************************************
    1587 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_range3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int x;
void func(void)
{

        if (x < 1)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (12 < x)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (x <= 23)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (34 <= x)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (x >= 45)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (56 >= x)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (x > 67)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (78 > x)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (89 == x)
                __smatch_value("x");
        else
                __smatch_value("x");

        if (100 != x)
                __smatch_value("x");
        else
                __smatch_value("x");

        return;
}
/*
 * check-name: smatch range comparison
```



```
 * check-command: smatch -I.. sm_range3.c
 *
 * check-output-start
sm_range3.c:8 func() x = s32min-0
sm_range3.c:10 func() x = 1-s32max
sm_range3.c:13 func() x = 13-s32max
sm_range3.c:15 func() x = s32min-12
sm_range3.c:18 func() x = s32min-23
sm_range3.c:20 func() x = 24-s32max
sm_range3.c:23 func() x = 34-s32max
sm_range3.c:25 func() x = s32min-33
sm_range3.c:28 func() x = 45-s32max
sm_range3.c:30 func() x = s32min-44
sm_range3.c:33 func() x = s32min-56
sm_range3.c:35 func() x = 57-s32max
sm_range3.c:38 func() x = 68-s32max
sm_range3.c:40 func() x = s32min-67
sm_range3.c:43 func() x = s32min-77
sm_range3.c:45 func() x = 78-s32max
sm_range3.c:48 func() x = 89
sm_range3.c:50 func() x = s32min-88,90-s32max
sm_range3.c:53 func() x = s32min-99,101-s32max
sm_range3.c:55 func() x = 100
 * check-output-end
 */
```

```
**********************************************************
     622 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_range4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b, c;

static int frob(void)
{
        if (a > 5) {
                __smatch_value("a");
                return;
        }
        if (b++ > 5) {
                __smatch_value("b");
                return;
        }
        if (++c > 5) {
                __smatch_value("c");
                return;
        }
        __smatch_value("a");
        __smatch_value("b");
        __smatch_value("c");
}


/*
 * check-name: Smatch Range #4
 * check-command: smatch -I.. sm_range4.c
 *
 * check-output-start
sm_range4.c:8 frob() a = 6-s32max
sm_range4.c:12 frob() b = 7-s32max
sm_range4.c:16 frob() c = 6-s32max
sm_range4.c:19 frob() a = s32min-5
sm_range4.c:20 frob() b = s32min-6
sm_range4.c:21 frob() c = s32min-5
 * check-output-end
 */
```

```
**********************************************************
     381 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_range5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int main(unsigned int x, unsigned int y)
{
        switch (x) {
        case 0 ... 9:
                __smatch_implied(x);
                break;
        default:
                __smatch_implied(x);
        }
}

/*
 * check-name: smatch range #5
 * check-command: smatch -I.. sm_range5.c
 *
 * check-output-start
sm_range5.c:7 main() implied: x = '0-9'
sm_range5.c:10 main() implied: x = '10-u32max'
 * check-output-end
 */
```

```
**********************************************************
     482 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_range6.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void func(void)
{
        long a = __smatch_rl("1-10+");
        long b = __smatch_rl("0,+");
        long c = __smatch_rl("10,23,45-+");

        __smatch_implied(a);
        __smatch_implied(b);
        __smatch_implied(c);
}

/*
 * check-name: smatch range #6
 * check-command: smatch -I.. sm_range6.c
 *
 * check-output-start
sm_range6.c:9 func() implied: a = '1-s64max'
sm_range6.c:10 func() implied: b = '0-s64max'
sm_range6.c:11 func() implied: c = '10,23,45-s64max'
 * check-output-end
 */
```

```
**********************************************************
     374 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_real_absolute1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"


void *p;
int min1, min2;
void func(unsigned long x)
{
        min1 = 18;
        min2 = (((unsigned char *)p)[12] + 8);
        if (min2 < min1)
                __smatch_implied(min2);
}

/*
 * check-name: Smatch real absolute #1
 * check-command: smatch -I.. sm_real_absolute1.c
 *
 * check-output-start
sm_real_absolute1.c:11 func() implied: min2 = '8-17'
 * check-output-end
 */
```

```
**********************************************************
    1762 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_rosenberg.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void memset(void *ptr, char c, int size){}

int copy_to_user(void *dest, void *data, int size){}
int some_func(struct foo *p){}

typedef struct zr364xx_pipeinfo {
        char x;
        int y;
} aa_policy_t;

struct aa_policy {
        int x;
};

struct foo {
        struct aa_policy a;
        int x;
        int y;
};

struct foo *p;
struct foo global_dec;
void *ptr;

int main(void)
{
        struct zr364xx_pipeinfo one;
        struct aa_policy two;
        aa_policy_t three;
        struct foo four;
        struct foo five;
        struct foo six;
        struct foo seven;
        struct foo eight;
        struct foo nine;

        p->a.x = 0;
        global_dec.x = 0;
        memset(&two, 0, sizeof(two));
        four.x = 0;
        six = five;
        some_func(&seven);
        eight.x = (four.x < 5 ? four.x : 5);
        eight.y = !five.y;
        if (some_func()) {
                nine.x = 1;
                nine.y = 2;
        }

        copy_to_user(ptr, &p->a, sizeof(struct aa_policy));
        copy_to_user(ptr, &global_dec, sizeof(global_dec));
        copy_to_user(ptr, &one, sizeof(one));
        copy_to_user(ptr, &two, sizeof(two));
        copy_to_user(ptr, &three, sizeof(three));
        copy_to_user(ptr, &four, sizeof(four));
        copy_to_user(ptr, &five, sizeof(five));
        copy_to_user(ptr, &six, sizeof(six));
        copy_to_user(ptr, &seven, sizeof(seven));
```



```
        copy_to_user(ptr, &eight, sizeof(eight));
        copy_to_user(ptr, &nine, sizeof(nine));
        return 0;
}
/*
 * check-name: Rosenberg Leaks
 * check-command: smatch -p=kernel -I.. sm_rosenberg.c
 *
 * check-output-start
sm_rosenberg.c:54 main() warn: check that 'one' doesn't leak information (struct
sm_rosenberg.c:56 main() warn: check that 'three' doesn't leak information (stru
sm_rosenberg.c:57 main() warn: check that 'four.y' doesn't leak information
sm_rosenberg.c:62 main() warn: check that 'nine.x' doesn't leak information
 * check-output-end
 */
```

```
**********************************************************
     739 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_select.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct foo {
   2         int a;
   3 };

   5 struct foo *a;
   6 struct foo *b;

   8 struct foo *c;
   9 struct foo *d;
  10 struct foo *e;
  11 void func (void)
  12 {
  13         if (a?b:0) {
  14                 a->a = 1;
  15                 b->a = 1;
  16         }
  17         a->a = 1;
  18         b->a = 1;
  19         e->a = 1;
  20         d = returns_nonnull();
  21         if (c?d:e) {
  22                 c->a = 1;
  23                 d->a = 1;
  24                 e->a = 1;
  25         }
  26         e->a = 1;
  27 }

  29 /*
  30  * check-name: Ternary Conditions
  31  * check-command: smatch sm_select.c
  32  *
  33  * check-output-start
  34 sm_select.c:17 func() error: we previously assumed 'a' could be null (see line 1
  35 sm_select.c:18 func() error: we previously assumed 'b' could be null (see line 1
  36 sm_select.c:21 func() warn: variable dereferenced before check 'e' (see line 19)
  37 sm_select.c:22 func() error: we previously assumed 'c' could be null (see line 2
  38  * check-output-end
  39  */
```

```
**********************************************************
    2446 Fri Dec 21 15:01:12 2018
new/usr/src/tools/smatch/src/validation/sm_select3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b, c;
int func(void)
{
        if (a ? b : c)
                __smatch_value("a");

        __smatch_note("Test #1 a ? 1 : c");
        if (a ? 1 : c) {
                __smatch_value("a");
                __smatch_value("c");
                if (!a)
                        __smatch_value("c");
                if (!c)
                        __smatch_value("a");
        } else {
                __smatch_value("a");
                __smatch_value("c");
        }

        __smatch_note("Test #2 a ? 0 : c");
        if (a ? 0 : c) {
                __smatch_value("a");
                __smatch_value("c");
                if (!a)
                        __smatch_value("c");
        } else {
                __smatch_value("a");
                __smatch_value("c");
                if (!a)
                        __smatch_value("c");
                if (!c)
                        __smatch_value("a");
        }

        __smatch_note("Test #3 a ? b : 1");
        if (a ? b : 1) {
                __smatch_value("a");
                __smatch_value("b");
                if (!a)
                        __smatch_value("b");
                if (!b)
                        __smatch_value("a");
        } else {
                __smatch_value("a");
                __smatch_value("b");
                if (!b)
                        __smatch_value("a");
        }

        __smatch_note("Test #2 a ? b : 0");
        if (a ? b : 0) {
                __smatch_value("a");
                __smatch_value("b");
        } else {
                __smatch_value("a");
                __smatch_value("b");
                if (a)
                        __smatch_value("b");
```



```
                if (b)
                        __smatch_value("a");
        }
}


/*
 * check-name: Ternary Conditions #3
 * check-command: smatch -I.. sm_select3.c
 *
 * check-output-start
sm_select3.c:7 func() a = s32min-s32max
sm_select3.c:9 func() Test #1 a ? 1 : c
sm_select3.c:11 func() a = s32min-s32max
sm_select3.c:12 func() c = s32min-s32max
sm_select3.c:14 func() c = s32min-(-1),1-s32max
sm_select3.c:16 func() a = s32min-(-1),1-s32max
sm_select3.c:18 func() a = 0
sm_select3.c:19 func() c = 0
sm_select3.c:22 func() Test #2 a ? 0 : c
sm_select3.c:24 func() a = 0
sm_select3.c:25 func() c = s32min-(-1),1-s32max
sm_select3.c:27 func() c = s32min-(-1),1-s32max
sm_select3.c:29 func() a = s32min-s32max
sm_select3.c:30 func() c = s32min-s32max
sm_select3.c:32 func() c = 0
sm_select3.c:34 func() a = s32min-s32max
sm_select3.c:37 func() Test #3 a ? b : 1
sm_select3.c:39 func() a = s32min-s32max
sm_select3.c:40 func() b = s32min-s32max
sm_select3.c:42 func() b = s32min-s32max
sm_select3.c:44 func() a = 0
sm_select3.c:46 func() a = s32min-(-1),1-s32max
sm_select3.c:47 func() b = 0
sm_select3.c:49 func() a = s32min-(-1),1-s32max
sm_select3.c:52 func() Test #2 a ? b : 0
sm_select3.c:54 func() a = s32min-(-1),1-s32max
sm_select3.c:55 func() b = s32min-(-1),1-s32max
sm_select3.c:57 func() a = s32min-s32max
sm_select3.c:58 func() b = s32min-s32max
sm_select3.c:60 func() b = 0
sm_select3.c:62 func() a = 0
 * check-output-end
 */
```

**********
445 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_select4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#include "check_debug.h"

int frob();

int a, b, c;
void func(unsigned long x)
{
        if (x >= 4)
                return;

        __smatch_value("x");
        if ((!(a) ? -19 : (((b && c) ? frob() : -515))))
                __smatch_value("x");
        __smatch_value("x");
}
/*
 * check-name: Smatch Ternary #4
 * check-command: smatch -I.. sm_select4.c
 *
 * check-output-start
sm_select4.c:11 func() x = 0-3
sm_select4.c:13 func() x = 0-3
sm_select4.c:14 func() x = 0-3
 * check-output-end
 */
```

```
**********************************************************
     512 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_select5.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int load_sig(unsigned long sig)
{
        return sig < 4 ? 0 : -12;
}

int a;
void test(void)
{
        int ret;

        ret = load_sig(a);
        if (ret) {
                __smatch_implied(ret);
                __smatch_implied(a);
        } else {
                __smatch_implied(a);
        }
}

/*
 * check-name: smatch select #5
 * check-command: smatch -I.. sm_select5.c
 *
 * check-output-start
sm_select5.c:15 test() implied: ret = '(-12)'
sm_select5.c:16 test() implied: a = 's32min-s32max'
sm_select5.c:18 test() implied: a = '0-3'
 * check-output-end
 */
```

```
**********************************************************
     658 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_select_assign.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

void frob();

#define min(a, b) ((a) < (b) ? (a) : (b))

void func(void)
{
        int i;
        int val;

        for (i = 0; i < 10; i++) {
                val = min(5, i);
                __smatch_value("val");
        }

        i++;
        __smatch_value("i");
        val = min(100, i);
        __smatch_value("val");

        for (i = 0; i < 10; i++)
                frob();

        val = min(100, i);
        __smatch_value("val");
}
/*
 * check-name: assigning select statements
 * check-command: smatch -I.. sm_select_assign.c
 *
 * check-output-start
sm_select_assign.c:14 func() val = 0-5
sm_select_assign.c:18 func() i = 11-s32max
sm_select_assign.c:20 func() val = 11-100
sm_select_assign.c:26 func() val = 10
 * check-output-end
 */
```

```
**********************************************************
    1043 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_skb.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct sk_buff {
        int valuable_information;
};
struct foo {
        int x;
};
struct ture {
        struct sk_buff *skb;
};

struct wrap1 {
        struct ture *a;
};
struct wrap2 {
        struct foo *c;
        struct wrap1 *b;
};
struct wrap3 {
        struct foo *c;
};

struct sk_buff *skb;
struct sk_buff **ptr;
struct ture *x;
struct ture xx;
struct wrap1 *u;
struct wrap2 *y;
struct wrap3 *z;

void kfree(void *data);

void func (void)
{
        kfree(skb);
        kfree(x->skb);
        kfree(xx.skb);
        kfree(y->c);
        kfree(u->a->skb);
        kfree(u->a);
        kfree(y->b->a->skb);
        kfree(z->c);
        kfree(ptr);
}
/*
 * check-name: kfree_skb() test
 * check-command: smatch -p=kernel sm_skb.c
 *
 * check-output-start
sm_skb.c:34 func() error: use kfree_skb() here instead of kfree(skb)
sm_skb.c:35 func() error: use kfree_skb() here instead of kfree(x->skb)
sm_skb.c:36 func() error: use kfree_skb() here instead of kfree(xx.skb)
sm_skb.c:38 func() error: use kfree_skb() here instead of kfree(u->a->skb)
sm_skb.c:40 func() error: use kfree_skb() here instead of kfree(y->b->a->skb)
 * check-output-end
 */
```

```
**********************************************************
    1270 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_skb2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

struct sk_buff {
        unsigned char *head, *data;
        unsigned short network_header;
};

struct foo {
        int a, b, c;
};

static inline unsigned char *skb_network_header(const struct sk_buff *skb)
{
        return skb->head + skb->network_header;
}

static inline int skb_network_offset(const struct sk_buff *skb)
{
        return skb_network_header(skb) - skb->data;
}

int frob(struct sk_buff *skb)
{
        struct foo *p;
        int x, y;

        __smatch_user_rl(*skb->data);
        __smatch_user_rl(skb->data + 1);
        __smatch_user_rl(*(int *)skb->data);
        __smatch_user_rl(skb->data - skb_network_header(skb));

        p = skb->data;
        x = *(int *)skb->data;
        y = skb->data[1];

        __smatch_user_rl(p->a);
        __smatch_user_rl(x);
        __smatch_user_rl(y);

        return 0;
}

/*
 * check-name: smatch: userdata from skb
 * check-command: smatch -p=kernel -I.. sm_skb2.c
 *
 * check-output-start
sm_skb2.c:27 frob() user rl: '*skb->data' = '0-255'
sm_skb2.c:28 frob() user rl: 'skb->data + 1' = ''
sm_skb2.c:29 frob() user rl: '*skb->data' = 's32min-s32max'
sm_skb2.c:30 frob() user rl: 'skb->data - skb_network_header(skb)' = ''
sm_skb2.c:36 frob() user rl: 'p->a' = 's32min-s32max'
sm_skb2.c:37 frob() user rl: 'x' = 's32min-s32max'
sm_skb2.c:38 frob() user rl: 'y' = '0-255'
 * check-output-end
 */
```

**********
462 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_skb3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
#include "check_debug.h"

struct sk_buff {
        unsigned char *head, *data;
        unsigned short network_header;
};

struct foo {
        int a, b, c;
};

int frob(struct sk_buff *skb)
{
        struct foo *p;

        p = skb->data + sizeof(int) * 2;
        __smatch_user_rl(p->a);

        return 0;
}

/*
 * check-name: smatch: userdata from skb #3
 * check-command: smatch -p=kernel -I.. sm_skb3.c
 *
 * check-output-start
sm_skb3.c:17 frob() user rl: 'p->a' = 's32min-s32max'
 * check-output-end
 */
```

```
**********************************************************
     428 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_strlen.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
  1 int strlen(char *str);
  2 int strcpy(char *str);
  3
  4 void func (char *input)
  5 {
  6         int input_len;
  7         char buf[4];
  8
  9         input_len = strlen(input);
 10         if (input_len <= 5) {
 11                 strcpy(buf, input);
 12         }
 13         if (input_len <= 3) {
 14                 strcpy(buf, input);
 15         }
 16 }
 17 /*
 18  * check-name: Smatch strlen test
 19  * check-command: smatch sm_strlen.c
 20  *
 21  * check-output-start
 22 sm_strlen.c:11 func() error: strcpy() 'input' too large for 'buf' (6 vs 4)
 23  * check-output-end
 24  */
```

```
**********************************************************
     672 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_strlen2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int strlen(char *str);
int strcpy(char *str);

void func (char *input1, char *input2, char *input3)
{
        char buf1[4];
        char buf2[4];
        char buf3[4];

        if (strlen(input1) > 4)
                return;
        strcpy(buf1, input1);

        if (10 > strlen(input2))
                strcpy(buf2, input2);

        if (strlen(input3) <= 4)
                strcpy(buf3, input3);
}
/*
 * check-name: Smatch strlen test #2
 * check-command: smatch sm_strlen2.c
 *
 * check-output-start
sm_strlen2.c:12 func() error: strcpy() 'input1' too large for 'buf1' (5 vs 4)
sm_strlen2.c:15 func() error: strcpy() 'input2' too large for 'buf2' (10 vs 4)
sm_strlen2.c:18 func() error: strcpy() 'input3' too large for 'buf3' (5 vs 4)
 * check-output-end
 */
```

```
**********************************************************
     427 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_strlen3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int strlen(const char *str);
int strnlen(const char *str, int limit);

int func(void)
{
        __smatch_implied(strlen("foo"));
        __smatch_implied(strnlen("foo", 2));
}

/*
 * check-name: Smatch strlen test #3
 * check-command: smatch -I.. sm_strlen3.c
 *
 * check-output-start
sm_strlen3.c:8 func() implied: strlen("foo") = '3'
sm_strlen3.c:9 func() implied: strnlen("foo", 2) = '2'
 * check-output-end
 */
```

**********
622 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_struct_assign1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 #include "check_debug.h"
 2
 3 void memcpy(void *dest, void *src, int size);
 4 void memset(void *dest, char c, int size);
 5
 6
 7 struct foo {
 8         int x, y;
 9 };
10
11 void test(void)
12 {
13         struct foo src = {1, 41};
14         struct foo dest;
15
16         memcpy(&dest, &src, sizeof(dest));
17         __smatch_implied(dest.x + dest.y);
18         memset(&dest, 0, sizeof(dest));
19         __smatch_implied(dest.x + dest.y);
20
21 }
22
23 /*
24  * check-name: smatch struct assignment #1
25  * check-command: smatch -I.. sm_struct_assign1.c
26  *
27  * check-output-start
28 sm_struct_assign1.c:17 test() implied: dest.x + dest.y = '42'
29 sm_struct_assign1.c:19 test() implied: dest.x + dest.y = '0'
30  * check-output-end
31  */
```

```
**********************************************************
     833 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_switch.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 //#include <stdlib.h>

   3 struct foo {
   4         int a;
   5 };

   7 struct foo *a;
   8 struct foo *b;
   9 struct foo *c;
  10 struct foo *d;
  11 int x;

  13 void func (void)
  14 {
  15         a = 0;
  16         b = 0;
  17         c = 0;
  18         d = 0;

  20         switch(x) {
  21         case 1:
  22                 a = returns_nonnull();
  23                 break;
  24         case 2:
  25                 b = returns_nonnull();
  26                 break;
  27         case 3:
  28                 c = returns_nonnull();
  29                 break;
  30         default:
  31                 d = returns_nonnull();
  32         }

  34         switch(x) {
  35         case 1:
  36                 a->a = 1;
  37         case 2:
  38                 a->a = 2;
  39                 b->a = 3;
  40                 break;
  41         case 3:
  42                 c->a = 4;
  43                 break;
  44         case 4:
  45                 d->a = 5;
  46                 break;
  47         }
  48 }
  49 /*
  50  * check-name: Smatch switch handling
  51  * check-command: smatch --spammy sm_switch.c
  52  * check-known-to-fail
  53  *
  54  * check-output-start
  55 sm_switch.c:38 func() warn: missing break? reassigning 'a->a'
  56 sm_switch.c:38 func() error: potential NULL dereference 'a'.
  57 sm_switch.c:39 func() error: potential NULL dereference 'b'.
  58  * check-output-end
  59  */
```

```
**********************************************************
     745 Fri Dec 21 15:01:13 2018
new/usr/src/tools/smatch/src/validation/sm_switch2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
struct foo {
        int a;
};

struct foo *a;
struct foo *b;
struct foo *c;
struct foo *d;
int x;

void func (void)
{
        a = 0;
        b = 0;
        c = 0;
        d = 0;

        if (x == 1)
                a = some_func();
        else if (x == 2)
                b = some_func();
        else if (x == 3)
                c = some_func();
        else
                d = some_func();

        switch(x) {
        case 1:
                a->a = 1;
        case 2:
                a->a = 2;
                b->a = 3;
                break;
        case 3:
                c->a = 4;
                break;
        case 4:
                d->a = 5;
                break;
        }
}
/*
 * check-name: Smatch switch handling #2
 * check-command: smatch --spammy sm_switch2.c
 *
 * check-output-start
sm_switch2.c:31 func() warn: missing break? reassigning 'a->a'
sm_switch2.c:31 func() error: potential NULL dereference 'a'.
sm_switch2.c:32 func() error: potential NULL dereference 'b'.
 * check-output-end
 */
```

```
**********************************************************
     588 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_switch3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int a, b;

int frob(void);

int test(int size)
{
        a = 0;

        if (({switch (frob()) {
                case 1:
                        a = 2;
                        break;
                default:
                        a = 3;
            }
            b;}))
                ;
        __smatch_implied(a);

        a = 4;

        if (({switch (2) {
                case 1:
                        a = 5;
                        break;
                case 2:
                        a = 6;
                        break;
                default:
                        a = 7;
            }
            b;}))
                ;
        __smatch_implied(a);

        return 0;
}

/*
 * check-name: smatch: switch #3
 * check-command: smatch -I.. sm_switch3.c
 *
 * check-output-start
sm_switch3.c:20 test() implied: a = '2-3'
sm_switch3.c:36 test() implied: a = '6'
 * check-output-end
 */
```

```
**********************************************************
     512 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_user_data1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int copy_from_user(void *dest, void *src, int size);

struct my_struct {
        int x, y;
};

void *pointer;

void copy_stuff(struct my_struct *foo)
{
        copy_from_user(foo, pointer, sizeof(*foo));
}

void test(void)
{
        struct my_struct foo;

        copy_stuff(&foo);
        __smatch_user_rl(foo.x);
}
/*
 * check-name: smatch user data #1
 * check-command: smatch -p=kernel -I.. sm_user_data1.c
 *
 * check-output-start
sm_user_data1.c:21 test() user rl: 'foo.x' = 's32min-s32max'
 * check-output-end
 */
```

```
**********************************************************
     549 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_user_data2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int copy_from_user(void *dest, void *src, int size){}

struct my_struct {
        int x, y;
};

void *pointer;
struct my_struct *dest;

struct my_struct *returns_copy(void)
{
        copy_from_user(dest, pointer, sizeof(*dest));
        return dest;
}

struct my_struct *a;
void test(void)
{
        a = returns_copy();
        __smatch_user_rl(a->x);
}

/*
 * check-name: smatch user data #2
 * check-command: smatch -p=kernel -I.. sm_user_data2.c
 *
 * check-output-start
sm_user_data2.c:22 test() user rl: 'a->x' = 's32min-s32max'
 * check-output-end
 */
```

```
**********************************************************
     733 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_user_data3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int copy_from_user(void *dest, void *src, int size){}

struct my_struct {
        int x, y;
};

struct my_struct *returns_filter(struct my_struct *p)
{
        return p;
}

struct my_struct *src, *a, *b;
void test(void)
{
        copy_from_user(a, src, sizeof(*a));
        b = returns_filter(a);
        __smatch_user_rl(b->y);
        b = returns_filter(src);
        __smatch_user_rl(b->y);
        b = returns_filter(a);
        __smatch_user_rl(b->y);
}

/*
 * check-name: smatch user data #3
 * check-command: smatch -p=kernel -I.. sm_user_data3.c
 *
 * check-output-start
sm_user_data3.c:19 test() user rl: 'b->y' = 's32min-s32max'
sm_user_data3.c:21 test() user rl: 'b->y' = ''
sm_user_data3.c:23 test() user rl: 'b->y' = 's32min-s32max'
 * check-output-end
 */
```

```
**********************************************************
     803 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_user_data4.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int copy_from_user(void *dest, void *src, int size);

struct ear {
        int x, y;
};

void *src;
int returns_user_data(void)
{
        int x;

        copy_from_user(&x, src, sizeof(int));
        return x;
}

struct ear *dest;
struct ear *returns_user_member(void)
{
        copy_from_user(&dest->x, src, sizeof(int));
        return dest;
}
void test(void)
{
        struct ear *p;
        int x;

        x = returns_user_data();
        __smatch_user_rl(x);
        p = returns_user_member();
        __smatch_user_rl(p);
        __smatch_user_rl(p->x);
}

/*
 * check-name: smatch user data #4
 * check-command: smatch -p=kernel -I.. sm_user_data4.c
 *
 * check-output-start
sm_user_data4.c:30 test() user rl: 'x' = 's32min-s32max'
sm_user_data4.c:32 test() user rl: 'p' = ''
sm_user_data4.c:33 test() user rl: 'p->x' = 's32min-s32max'
 * check-output-end
 */
```

```
**********************************************************
     310 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_val_parse1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#include "check_debug.h"

int main(int x)
{
        x = __smatch_type_rl(int, "s32min-s32max[$2 + 4]", 5);
        __smatch_implied(x);

        return 0;
}
/*
 * check-name: smatch parse value
 * check-command: smatch -I.. sm_val_parse1.c
 *
 * check-output-start
sm_val_parse1.c:6 main() implied: x = '9'
 * check-output-end
 */
```

```
**********************************************************
     464 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_wine_filehandles.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 void * CreateFile();
   2 void * socket();

   4 int func (void)
   5 {
   6         int *x;

   8         if (x = CreateFile()) {
   9
  10         }

  12         x = socket();
  13         if (x != 0) {
  14
  15         }
  16         return;
  17 }
  18 /*
  19  * check-name: use INVALID_HANDLE_VALUE not zero
  20  * check-command: smatch -p=wine sm_wine_filehandles.c
  21  *
  22  * check-output-start
  23 sm_wine_filehandles.c:8 func() error: comparing a filehandle against zero 'x'
  24 sm_wine_filehandles.c:13 func() error: comparing a filehandle against zero 'x'
  25  * check-output-end
  26  */
```

```
**********************************************************
    1036 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/sm_wine_locking.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
int create_window_handle(int x);
void WIN_ReleasePtr(int x);
void EnterCriticalSection(int x);
void LeaveCriticalSection(int x);
void USER_Lock(void);
void USER_Unlock(void);
int GDI_GetObjPtr(int x);
void GDI_ReleaseObj(int x);

int a, b, c, d, e, z;

void test1(void)
{
        b = create_window_handle(a);
        z = frob();

        if (d = GDI_GetObjPtr(e))
                GDI_ReleaseObj(e);
        if (GDI_GetObjPtr(e))
                GDI_ReleaseObj(e);
        EnterCriticalSection(c);
        USER_Lock();
        if (b) {
                LeaveCriticalSection(c);
                WIN_ReleasePtr(b);
        }
        WIN_ReleasePtr(b);
        if (z)
                return;
        USER_Unlock();
        if (!b)
                LeaveCriticalSection(c);
}
/*
 * check-name: WINE locking
 * check-command: smatch -p=wine --spammy sm_wine_locking.c
 *
 * check-output-start
sm_wine_locking.c:27 test1() error: double unlock 'create_window_handle:b'
sm_wine_locking.c:29 test1() warn: 'CriticalSection:c' is sometimes locked here
sm_wine_locking.c:32 test1() warn: inconsistent returns 'USER_Lock:'.
  Locked on:   line 29
  Unlocked on: line 32
 * check-output-end
 */
```

```
**********************************************************
      96 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/smatch_db_test.sh
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 #!/bin/bash

   3 rm -f smatch_db.sqlite

   5 ./build_smatch_db.sh $*
   6 ../smatch $*

   8 rm smatch_db.sqlite
```

```
**********************************************************
    1675 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/specifiers1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static void OK(void)
{
#define TEST(x) { T a; x *b = &a; }
#define TEST2(x, y) TEST(x y) TEST(y x)
#define TEST3(x, y, z) TEST(x y z) TEST(x z y) TEST(y x z) \
                       TEST(y z x) TEST(z x y) TEST(z y x)
#define TEST4(x, y, z, w) TEST2(x y, z w) TEST2(x y, w z) \
                          TEST2(y x, z w) TEST2(y x, w z) \
                          TEST2(x z, y w) TEST2(x z, w y) \
                          TEST2(z x, y w) TEST2(z x, w y) \
                          TEST2(x w, y z) TEST2(x w, z y) \
                          TEST2(w x, y z) TEST2(w x, z y)


#define T char
TEST(char)
#undef T

#define T signed char
TEST2(char, signed)
#undef T

#define T unsigned char
TEST2(char, unsigned)
#undef T

#define T short
TEST(short)
TEST2(int, short)
#undef T

#define T int
TEST(int)
#undef T

#define T long
TEST(long)
TEST2(int, long)
#undef T

#define T long long
TEST2(long, long)
TEST3(int, long, long)
#undef T

#define T signed short
TEST2(short, signed)
TEST3(int, short, signed)
#undef T

#define T signed
TEST(signed)
TEST2(int, signed)
#undef T

#define T signed long
TEST2(long, signed)
TEST3(int, long, signed)
#undef T
```



```
#define T signed long long
TEST3(long, long, signed)
TEST4(int, long, long, signed)
#undef T

#define T unsigned short
TEST2(short, unsigned)
TEST3(int, short, unsigned)
#undef T

#define T unsigned
TEST(unsigned)
TEST2(int, unsigned)
#undef T

#define T unsigned long
TEST2(long, unsigned)
TEST3(int, long, unsigned)
#undef T

#define T unsigned long long
TEST3(long, long, unsigned)
TEST4(int, long, long, unsigned)
#undef T

#define T float
TEST(float)
#undef T

#define T double
TEST(double)
#undef T

#define T long double
TEST2(double, long)
#undef T
}
/*
 * check-name: valid specifier combinations
 * check-command: sparse $file
 */
```

```
**********************************************************
    6766 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/specifiers2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef int T;
void BAD(
char char,
char int,
char double,
char float,
char long,
char short,
int char,
int int,
int double,
int float,
double char,
double int,
double double,
double float,
double short,
double signed,
double unsigned,
float char,
float int,
float double,
float float,
float short,
float long,
float signed,
float unsigned,
short char,
short double,
short float,
short short,
short long,
long char,
long float,
long short,
signed double,
signed float,
signed signed,
signed unsigned,
unsigned double,
unsigned float,
unsigned signed,
unsigned unsigned,
unsigned signed,
long long long,
long double long,
long long double,
double long long,
T char,
T int,
T double,
T float,
T short,
T long,
T signed,
T unsigned,
T void,
void char,
void int,
void double,
```



```
void float,
void short,
void long,
void signed,
void unsigned,
char void,
int void,
double void,
float void,
short void,
long void,
signed void,
unsigned void,
void void
);
/*
 * check-name: invalid specifier combinations
 * check-error-start
specifiers2.c:3:6: error: two or more data types in declaration specifiers
specifiers2.c:4:6: error: two or more data types in declaration specifiers
specifiers2.c:5:6: error: two or more data types in declaration specifiers
specifiers2.c:6:6: error: two or more data types in declaration specifiers
specifiers2.c:7:6: error: impossible combination of type specifiers: char long
specifiers2.c:8:6: error: impossible combination of type specifiers: char short
specifiers2.c:9:5: error: two or more data types in declaration specifiers
specifiers2.c:10:5: error: two or more data types in declaration specifiers
specifiers2.c:11:5: error: two or more data types in declaration specifiers
specifiers2.c:12:5: error: two or more data types in declaration specifiers
specifiers2.c:13:8: error: two or more data types in declaration specifiers
specifiers2.c:14:8: error: two or more data types in declaration specifiers
specifiers2.c:15:8: error: two or more data types in declaration specifiers
specifiers2.c:16:8: error: two or more data types in declaration specifiers
specifiers2.c:17:8: error: impossible combination of type specifiers: double sho
specifiers2.c:18:8: error: impossible combination of type specifiers: double sig
specifiers2.c:19:8: error: impossible combination of type specifiers: double uns
specifiers2.c:20:7: error: two or more data types in declaration specifiers
specifiers2.c:21:7: error: two or more data types in declaration specifiers
specifiers2.c:22:7: error: two or more data types in declaration specifiers
specifiers2.c:23:7: error: two or more data types in declaration specifiers
specifiers2.c:24:7: error: impossible combination of type specifiers: float shor
specifiers2.c:25:7: error: impossible combination of type specifiers: float long
specifiers2.c:26:7: error: impossible combination of type specifiers: float sign
specifiers2.c:27:7: error: impossible combination of type specifiers: float unsi
specifiers2.c:28:7: error: impossible combination of type specifiers: short char
specifiers2.c:29:7: error: impossible combination of type specifiers: short doub
specifiers2.c:30:7: error: impossible combination of type specifiers: short floa
specifiers2.c:31:7: error: impossible combination of type specifiers: short shor
specifiers2.c:32:7: error: impossible combination of type specifiers: short long
specifiers2.c:33:6: error: impossible combination of type specifiers: long char
specifiers2.c:34:6: error: impossible combination of type specifiers: long float
specifiers2.c:35:6: error: impossible combination of type specifiers: long short
specifiers2.c:36:8: error: impossible combination of type specifiers: signed dou
specifiers2.c:37:8: error: impossible combination of type specifiers: signed flo
specifiers2.c:38:8: error: impossible combination of type specifiers: signed sig
specifiers2.c:39:8: error: impossible combination of type specifiers: signed uns
specifiers2.c:40:10: error: impossible combination of type specifiers: unsigned
specifiers2.c:41:10: error: impossible combination of type specifiers: unsigned
specifiers2.c:42:10: error: impossible combination of type specifiers: unsigned
specifiers2.c:43:10: error: impossible combination of type specifiers: unsigned
specifiers2.c:44:10: error: impossible combination of type specifiers: unsigned
specifiers2.c:45:11: error: impossible combination of type specifiers: long long
specifiers2.c:46:13: error: impossible combination of type specifiers: long long
specifiers2.c:47:11: error: impossible combination of type specifiers: long long
specifiers2.c:48:13: error: impossible combination of type specifiers: long long
specifiers2.c:49:3: error: two or more data types in declaration specifiers
specifiers2.c:50:3: error: two or more data types in declaration specifiers
```

```
specifiers2.c:51:3: error: two or more data types in declaration specifiers
specifiers2.c:52:3: error: two or more data types in declaration specifiers
specifiers2.c:53:3: error: two or more data types in declaration specifiers
specifiers2.c:54:3: error: two or more data types in declaration specifiers
specifiers2.c:55:3: error: two or more data types in declaration specifiers
specifiers2.c:56:3: error: two or more data types in declaration specifiers
specifiers2.c:57:3: error: two or more data types in declaration specifiers
specifiers2.c:58:6: error: two or more data types in declaration specifiers
specifiers2.c:59:6: error: two or more data types in declaration specifiers
specifiers2.c:60:6: error: two or more data types in declaration specifiers
specifiers2.c:61:6: error: two or more data types in declaration specifiers
specifiers2.c:62:6: error: two or more data types in declaration specifiers
specifiers2.c:63:6: error: two or more data types in declaration specifiers
specifiers2.c:64:6: error: two or more data types in declaration specifiers
specifiers2.c:65:6: error: two or more data types in declaration specifiers
specifiers2.c:66:6: error: two or more data types in declaration specifiers
specifiers2.c:67:5: error: two or more data types in declaration specifiers
specifiers2.c:68:8: error: two or more data types in declaration specifiers
specifiers2.c:69:7: error: two or more data types in declaration specifiers
specifiers2.c:70:7: error: impossible combination of type specifiers: short void
specifiers2.c:71:6: error: impossible combination of type specifiers: long void
specifiers2.c:72:8: error: impossible combination of type specifiers: signed voi
specifiers2.c:73:10: error: impossible combination of type specifiers: unsigned
specifiers2.c:74:6: error: two or more data types in declaration specifiers
 * check-error-end
 */
```

**********
226 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/static-forward-decl.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 static int f(void);
 2
 3 int f(void)
 4 {
 5         return 0;
 6 }
 7 /*
 8  * check-name: static forward declaration
 9  *
10  * check-error-start
11 static-forward-decl.c:3:5: warning: symbol 'f' was not declared. Should it be st
12  * check-error-end
13  */
```

```
**********************************************************
    1688 Fri Dec 21 15:01:14 2018
new/usr/src/tools/smatch/src/validation/static_assert.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
_Static_assert(1, "global ok");

struct foo {
        _Static_assert(1, "struct ok");
};

void bar(void)
{
        _Static_assert(1, " func1 ok");
        int i;
        i = 0;
        _Static_assert(1, " func2 ok");

        if (1) {
                _Static_assert(1, " func3 ok");
        }
}

_Static_assert(0, "expected assertion failure");

static int f;
_Static_assert(f, "non-constant expression");

static int *p;
_Static_assert(p, "non-integer expression");

_Static_assert(0.1, "float expression");

_Static_assert(!0 == 1, "non-trivial expression");

static char array[4];
_Static_assert(sizeof(array) == 4, "sizeof expression");

static const char non_literal_string[] = "non literal string";
_Static_assert(0, non_literal_string);

_Static_assert(1 / 0, "invalid expression: should not show up?");

struct s {
        char arr[16];
        _Static_assert(1, "inside struct");
};

union u {
        char c;
        int  i;
        _Static_assert(1, "inside union");
};

_Static_assert(sizeof(struct s) == 16, "sizeof assertion");

_Static_assert(1, );
_Static_assert(, "");
_Static_assert(,);

/*
 * check-name: static assertion
 *
 * check-error-start
static_assert.c:19:16: error: static assertion failed: "expected assertion failu
```



```
static_assert.c:22:16: error: bad constant expression
static_assert.c:25:16: error: bad constant expression
static_assert.c:27:16: error: bad constant expression
static_assert.c:35:19: error: bad or missing string literal
static_assert.c:37:18: error: bad constant expression
static_assert.c:52:19: error: bad or missing string literal
static_assert.c:53:16: error: Expected constant expression
static_assert.c:54:16: error: Expected constant expression
static_assert.c:54:17: error: bad or missing string literal
 * check-error-end
 */
```

```
**********************************************************
     162 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/strict-prototypes0.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 extern void func1();
2 extern void myfunction(), myfunc2();

4 /*
5  * check-name: strict-prototypes disabled
6  * check-command: sparse -Wno-strict-prototypes $file
7  */
```

```
**********************************************************
     475 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/strict-prototypes1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 extern void func0();
   2 extern void func1(), func2();

   4 /*
   5  * check-name: strict-prototypes enabled
   6  * check-command: sparse -Wstrict-prototypes $file
   7  * check-known-to-fail
   8  *
   9  * check-error-start
  10 strict-prototypes1.c:1:18: warning: non-ANSI function declaration of function 'f
  11 strict-prototypes1.c:2:18: warning: non-ANSI function declaration of function 'f
  12 strict-prototypes1.c:2:27: warning: non-ANSI function declaration of function 'f
  13  * check-error-end
  14  */
```

**********
306 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/struct-as.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 /*
 2  * Structure members should get the address
 3  * space of their pointer.
 4  */
 5 #define __user __attribute__((address_space(1)))

 7 struct hello {
 8         int a;
 9 };

11 extern int test(int __user *ip);

13 static int broken(struct hello __user *sp)
14 {
15         test(&sp->a);
16 }
17 /*
18  * check-name: Address space of a struct member
19  */
```

```
**********************************************************
     109 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/struct-attribute-placement.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 struct __attribute__((__aligned__(16))) foo {
2     int a;
3 };
4 /*
5  * check-name: struct attribute placement
6  */
```

**********
463 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/struct-ns1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********

```
 1 // This actually isn't allowed in C99, but sparse and gcc will take it:
 2 enum Foo;

 4 static void
 5 f (void)
 6 {
 7   enum Foo *pefoo;         // Pointer to incomplete type
 8   struct Foo;              // Forward declaration
 9   struct Foo *psfoo;       // Pointer to incomplete type
10   {
11     struct Foo { int foo; }; // Local definition.
12     struct Foo foo;          // variable declaration.
13     foo.foo = 1;
14   }
15 }

17 enum Foo { FOO };
18 /*
19  * check-name: struct namespaces #1
20  */
```

```
**********************************************************
     473 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/struct-ns2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static void
g (struct Bar { int i; } *x)
{
  struct Bar y;
  y.i = 1;
}

static void
h (void)
{
  // This is not in scope and should barf loudly.
  struct Bar y;
  y.i = 1;
}

/*
 * check-name: struct not in scope
 * check-known-to-fail
 *
 * check-error-start
struct-ns2.c:2:11: warning: bad scope for 'struct Bar'
struct-ns2.c:12:14: error: incomplete type/unknown size for 'y'
struct-ns2.c:13:5: error: using member 'i' in incomplete 'struct Bar'
 * check-error-end
 */
```

```
**********************************************************
     301 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/struct-size1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 struct A;
   2 struct B {
   3   struct A *pA;
   4 };
   5 struct C;
   6 struct E {
   7   struct A **pA;
   8   struct C *pC;
   9 };
  10 static void f(struct E *pE, struct B *pB)
  11 {
  12   pB->pA = pE->pA[0];
  13 }
  14 static const struct { int x; } foo[] = {{ 1 }};
  15 struct C {
  16   int bar[(sizeof foo/sizeof foo[0])];
  17 };

  19 /*
  20  * check-name: struct size
  21  */
```

```
**********************************************************
    1543 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/tautological-compare.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 typedef unsigned int u32;

 3 int seq(int a) { return a == a; }
 4 int sne(int a) { return a != a; }
 5 int slt(int a) { return a < a; }
 6 int sgt(int a) { return a > a; }
 7 int sle(int a) { return a <= a; }
 8 int sge(int a) { return a >= a; }

10 u32 ueq(u32 a) { return a == a; }
11 u32 une(u32 a) { return a != a; }
12 u32 ult(u32 a) { return a < a; }
13 u32 ugt(u32 a) { return a > a; }
14 u32 ule(u32 a) { return a <= a; }
15 u32 uge(u32 a) { return a >= a; }

17 /*
18  * check-name: tautological-compare
19  * check-command: sparse -Wno-decl -Wtautological-compare $file
20  *
21  * check-error-start
22 tautological-compare.c:3:30: warning: self-comparison always evaluates to true
23 tautological-compare.c:4:30: warning: self-comparison always evaluates to false
24 tautological-compare.c:5:29: warning: self-comparison always evaluates to false
25 tautological-compare.c:6:29: warning: self-comparison always evaluates to false
26 tautological-compare.c:7:30: warning: self-comparison always evaluates to true
27 tautological-compare.c:8:30: warning: self-comparison always evaluates to true
28 tautological-compare.c:10:30: warning: self-comparison always evaluates to true
29 tautological-compare.c:11:30: warning: self-comparison always evaluates to false
30 tautological-compare.c:12:29: warning: self-comparison always evaluates to false
31 tautological-compare.c:13:29: warning: self-comparison always evaluates to false
32 tautological-compare.c:14:30: warning: self-comparison always evaluates to true
33 tautological-compare.c:15:30: warning: self-comparison always evaluates to true
34  * check-error-end
35  */
```

```
**********************************************************
     577 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/test-be.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 int printf(char *c, ...);
   2 void exit(int c);

   4 #undef PRINT_OUTPUTS

   6 static void test_func_args(int x, int y)
   7 {
   8         if (x == y)
   9                 exit(1);
  10 }

  12 static int binop_s32(int x, int y)
  13 {
  14         int a;

  16         a = a + x;
  17         a = a / y;
  18         a = a * x;
  19         a = a - y;

  21         return a;
  22 }

  24 static void test_binops(void)
  25 {
  26         int tmp_s32 = binop_s32(987123, 234);

  28 #ifdef PRINT_OUTPUTS
  29         printf("binop_s32(987123, 234) == %d\n", tmp_s32);
  30 #else
  31         if (tmp_s32 != -1470599007)
  32                 exit(2);
  33 #endif
  34 }

  36 int main (int argc, char *argv[])
  37 {
  38         test_func_args(1, 2);
  39         test_binops();

  41         return 0;
  42 }

  44 /*
  45  * check-name: binary operations
  46  */
```

```
**********************************************************
    8911 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/test-suite
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#!/bin/sh

#set -x

cd $(dirname "$0")

default_path=".."
default_cmd="sparse \$file"
tests_list='find . -name '*.c' | sed -e 's#^\./\(.*\)#\1#' | sort'
prog_name='basename $0'

if [ ! -x "$default_path/sparse-llvm" ]; then
        disabled_cmds="sparsec sparsei sparse-llvm"
fi

# flags:
#       - some tests gave an unexpected result
failed=0

# counts:
#       - tests that have not been converted to test-suite format
#       - tests that are disabled
#       - tests that passed
#       - tests that failed
#       - tests that failed but are known to fail
unhandled_tests=0
disabled_tests=0
ok_tests=0
ko_tests=0
known_ko_tests=0

# defaults to not verbose
[ -z "$V" ] && V=0
[ $V -eq 0 ] && quiet=1 || quiet=0

##
# get_tag_value(file) - get the 'check-<...>' tags & values
get_tag_value()
{
        check_name=""
        check_command="$default_cmd"
        check_exit_value=0
        check_timeout=0
        check_known_to_fail=0
        check_error_ignore=0
        check_output_ignore=0
        check_output_contains=0
        check_output_excludes=0
        check_output_pattern=0

        lines=$(grep 'check-[a-z-]*' $1 | \
                sed -e 's/^.*\(check-[a-z-]*:*\) *\(.*\)$/\1 \2/')

        while read tag_val; do
                #echo "-> tag: '$tag'"
                #echo "-> val: '$val'"
                case $tag in
                check-name:)            check_name="$val" ;;
                check-command:)         check_command="$val" ;;
                check-exit-value:)      check_exit_value="$val" ;;
```



```
                check-timeout:)         [ -z "$val" ] && val=1
                                        check_timeout="$val" ;;
                check-known-to-fail)    check_known_to_fail=1 ;;
                check-error-ignore)     check_error_ignore=1 ;;
                check-output-ignore)    check_output_ignore=1 ;;
                check-output-contains:) check_output_contains=1 ;;
                check-output-excludes:) check_output_excludes=1 ;;
                check-output-pattern-)  check_output_pattern=1 ;;
                esac
        done << EOT
        $lines
EOT
}

##
# helper for has_(each|none)_patterns()
has_patterns()
{
        ifile="$1"
        patt="$2"
        ofile="$3"
        cmp="$4"
        grep "$patt:" "$ifile" | \
        sed -e "s/^.*$patt: *\(.*\)$/\1/" | \
        while read val; do
                grep -s -q "$val" "$ofile"
                if [ "$?" $cmp 0 ]; then
                        return 1
                fi
        done

        return $?
}

##
# has_each_patterns(ifile tag ofile) - does ofile contains some
#                         of the patterns given by ifile's tags?
#
# returns 0 if all present, 1 otherwise
has_each_patterns()
{
        has_patterns "$1" "$2" "$3" -ne
}

##
# has_none_patterns(ifile tag ofile) - does ofile contains some
#                         of the patterns given by ifile's tags?
#
# returns 1 if any present, 0 otherwise
has_none_patterns()
{
        has_patterns "$1" "$2" "$3" -eq
}

##
# nbr_patterns(ifile tag ofile) - does ofile contains the
#                         the patterns given by ifile's tags
#                         the right number of time?
nbr_patterns()
{
        ifile="$1"
        patt="$2"
        ofile="$3"
        grep "$patt-[0-9][0-9]*-times:" "$ifile" | \
        sed -e "s/^.*$patt-\([0-9][0-9]*\)-times: *\(.*\)/\1 \2/" | \
        while read nbr pat; do
```

```
127                 n=$(grep -s "$pat" "$ofile" | wc -l)
128                 if [ "$n" -ne "$nbr" ]; then
129                         return 1
130                 fi
131         done

133         return $?
134 }

136 ##
137 # verbose(string) - prints string if we are in verbose mode
138 verbose()
139 {
140         [ "$V" -eq "1" ] && echo "        $1"
141         return 0
142 }

144 ##
145 # error(string[, die]) - prints an error and exits with value die if given
146 error()
147 {
148         [ "$quiet" -ne 1 ] && echo "error: $1"
149         [ -n "$2" ] && exit $2
150         return 0
151 }

153 do_usage()
154 {
155 echo "$prog_name - a tiny automatic testing script"
156 echo "Usage: $prog_name [command] [command arguments]"
157 echo
158 echo "commands:"
159 echo "    none                        runs the whole test suite"
160 echo "    single file                 runs the test in 'file'"
161 echo "    format file [name [cmd]]    helps writing a new test case using cmd"
162 echo
163 echo "    help                        prints usage"
164 }

166 ##
167 # do_test(file) - tries to validate a test case
168 #
169 # it "parses" file, looking for check-* tags and tries to validate
170 # the test against an expected result
171 # returns:
172 #       - 0 if the test passed,
173 #       - 1 if it failed,
174 #       - 2 if it is not a "test-suite" test.
175 #       - 3 if the test is disabled.
176 do_test()
177 {
178         test_failed=0
179         file="$1"

181         get_tag_value $file

183         # can this test be handled by test-suite ?
184         # (it has to have a check-name key in it)
185         if [ "$check_name" = "" ]; then
186                 echo "warning: test '$file' unhandled"
187                 unhandled_tests=$(($unhandled_tests + 1))
188                 return 2
189         fi
190         test_name="$check_name"

192         # does the test provide a specific command ?
```



```
193         if [ "$check_command" = "" ]; then
194                 check_command="$defaut_command"
195         fi

197         # check for disabled commands
198         set -- $check_command
199         base_cmd=$1
200         for i in $disabled_cmds; do
201                 if [ "$i" = "$base_cmd" ] ; then
202                         disabled_tests=$(($disabled_tests + 1))
203                         echo "     DISABLE $test_name ($file)"
204                         return 3
205                 fi
206         done

208         cmd='eval echo $default_path/$check_command'

210         echo "     TEST    $test_name ($file)"

212         verbose "Using command       : $cmd"

214         # grab the expected exit value
215         expected_exit_value=$check_exit_value
216         verbose "Expecting exit value: $expected_exit_value"

218         # do we want a timeout?
219         if [ $check_timeout -ne 0 ]; then
220                 cmd="timeout -k 1s $check_timeout $cmd"
221         fi

223         # grab the actual output & exit value
224         $cmd 1> $file.output.got 2> $file.error.got
225         actual_exit_value=$?

227         must_fail=$check_known_to_fail
228         quiet=0
229         [ $must_fail -eq 1 ] && [ $V -eq 0 ] && quiet=1
230         known_ko_tests=$(($known_ko_tests + $must_fail))

232         for stream in output error; do
233                 eval ignore=\$check_${stream}_ignore
234                 [ $ignore -eq 1 ] && continue

236                 # grab the expected output
237                 sed -n "/check-$stream-start/,/check-$stream-end/p" $file \
238                 | grep -v check-$stream > "$file".$stream.expected

240                 diff -u "$file".$stream.expected "$file".$stream.got > "$file".$
241                 if [ "$?" -ne "0" ]; then
242                         error "actual $stream text does not match expected $stre
243                         error  "see $file.$stream.* for further investigation."
244                         [ $quiet -ne 1 ] && cat "$file".$stream.diff
245                         test_failed=1
246                 fi
247         done

249         if [ "$actual_exit_value" -ne "$expected_exit_value" ]; then
250                 error "Actual exit value does not match the expected one."
251                 error "expected $expected_exit_value, got $actual_exit_value."
252                 test_failed=1
253         fi

255         # verify the 'check-output-contains/excludes' tags
256         if [ $check_output_contains -eq 1 ]; then
257                 has_each_patterns "$file" 'check-output-contains' $file.output.g
258                 if [ "$?" -ne "0" ]; then
```

```
259                         error "Actual output doesn't contain some of the expecte
260                         test_failed=1
261                 fi
262         fi
263         if [ $check_output_excludes -eq 1 ]; then
264                 has_none_patterns "$file" 'check-output-excludes' $file.output.g
265                 if [ "$?" -ne "0" ]; then
266                         error "Actual output contains some patterns which are no
267                         test_failed=1
268                 fi
269         fi
270         if [ $check_output_pattern -eq 1 ]; then
271                 # verify the 'check-output-pattern-X-times' tags
272                 nbr_patterns "$file" 'check-output-pattern' $file.output.got
273                 if [ "$?" -ne "0" ]; then
274                         error "Actual output doesn't contain the pattern the exp
275                         test_failed=1
276                 fi
277         fi

279         [ "$test_failed" -eq "$must_fail" ] || failed=1

281         if [ "$must_fail" -eq "1" ]; then
282                 if [ "$test_failed" -eq "1" ]; then
283                         echo "info: test '$file' is known to fail"
284                 else
285                         echo "error: test '$file' is known to fail but succeed!"
286                         test_failed=1
287                 fi
288         fi

290         if [ "$test_failed" -eq "1" ]; then
291                 ko_tests=$(($ko_tests + 1))
292         else
293                 ok_tests=$(($ok_tests + 1))
294                 rm -f $file.{error,output}.{expected,got,diff}
295         fi
296         return $test_failed
297 }

299 do_test_suite()
300 {
301         for i in $tests_list; do
302                 do_test "$i"
303         done

305         # prints some numbers
306         tests_nr=$(($ok_tests + $ko_tests))
307         echo -n "Out of $tests_nr tests, $ok_tests passed, $ko_tests failed"
308         echo " ($known_ko_tests of them are known to fail)"
309         if [ "$unhandled_tests" -ne "0" ]; then
310                 echo "$unhandled_tests tests could not be handled by $prog_name"
311         fi
312         if [ "$disabled_tests" -ne "0" ]; then
313                 echo "$disabled_tests tests were disabled"
314         fi
315 }

317 ##
318 # do_format(file[, name[, cmd]]) - helps a test writer to format test-suite tags
319 do_format()
320 {
321         if [ -z "$2" ]; then
322                 fname="$1"
323                 fcmd=$default_cmd
324         elif [ -z "$3" ]; then
```



```
325                 fname="$2"
326                 fcmd=$default_cmd
327         else
328                 fname="$2"
329                 fcmd="$3"
330         fi
331         file="$1"
332         cmd='eval echo $default_path/$fcmd'
333         $cmd 1> $file.output.got 2> $file.error.got
334         fexit_value=$?
335         cat <<_EOF
336 /*
337  * check-name: $fname
338 _EOF
339         if [ "$fcmd" != "$default_cmd" ]; then
340                 echo " * check-command: $fcmd"
341         fi
342         if [ "$fexit_value" -ne "0" ]; then
343                 echo " * check-exit-value: $fexit_value"
344         fi
345         for stream in output error; do
346                 if [ -s "$file.$stream.got" ]; then
347                         echo " *"
348                         echo " * check-$stream-start"
349                         cat "$file.$stream.got"
350                         echo " * check-$stream-end"
351                 fi
352         done
353         echo " */"
354         return 0
355 }

357 ##
358 # arg_file(filename) - checks if filename exists
359 arg_file()
360 {
361         [ -z "$1" ] && {
362                 do_usage
363                 exit 1
364         }
365         [ -e "$1" ] || {
366                 error "Can't open file $1"
367                 exit 1
368         }
369         return 0
370 }

372 case "$1" in
373         '')
374                 do_test_suite
375                 ;;
376         single)
377                 arg_file "$2"
378                 do_test "$2"
379                 case "$?" in
380                         0) echo "$2 passed !";;
381                         1) echo "$2 failed !";;
382                         2) echo "$2 can't be handled by $prog_name";;
383                 esac
384                 ;;
385         format)
386                 arg_file "$2"
387                 do_format "$2" "$3" "$4"
388                 ;;
389         help | *)
390                 do_usage
```

```
391                 exit 1
392                 ;;
393 esac

395 exit $failed
```

```
**********************************************************
     161 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/testsuite-selfcheck1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 good

   3 /*
   4  * check-name: selfcheck1
   5  * check-command: sparse -E $file
   6  * check-output-ignore
   7  *
   8  * check-output-contains: good
   9  * check-output-excludes: evil
  10  */
```

```
**********************************************************
     153 Fri Dec 21 15:01:15 2018
new/usr/src/tools/smatch/src/validation/testsuite-selfcheck2.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 evil

   3 /*
   4  * check-name: selfcheck2
   5  * check-command: sparse -E $file
   6  * check-output-ignore
   7  * check-known-to-fail
   8  *
   9  * check-output-contains: good
  10  */
```

```
**********************************************************
     153 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/testsuite-selfcheck3.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 evil

   3 /*
   4  * check-name: selfcheck3
   5  * check-command: sparse -E $file
   6  * check-output-ignore
   7  * check-known-to-fail
   8  *
   9  * check-output-excludes: evil
  10  */
```

```
**********************************************************
     324 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/transparent-union.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
 1 struct a {
 2         int field;
 3 };
 4 struct b {
 5         int field;
 6 };

 8 typedef union {
 9         struct a *a;
10         struct b *b;
11 } transparent_arg __attribute__((__transparent_union__));

13 static void foo(transparent_arg arg)
14 {
15 }

17 static void bar(void)
18 {
19         struct b arg = { 0 };
20         foo((struct a *) &arg);
21 }

23 /*
24  * check-name: Transparent union attribute.
25  */
```

```
**********************************************************
     442 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/type-attribute-align.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __aligned(N)    __attribute__((aligned(N)))
#define alignof(X)      __alignof__(X)

struct s {
        short a, b, c;
} __aligned(2*sizeof(short));

static int fs(void) { return  sizeof(struct s); }
static int fa(void) { return alignof(struct s); }

void main(void)
{
        _Static_assert( sizeof(struct s) == 4 * sizeof(short), "size");
        _Static_assert(alignof(struct s) == 2 * sizeof(short), "alignment");
}

/*
 * check-name: type-attribute-align
 */
```

```
**********************************************************
     747 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/type-attribute-as.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __user          __attribute__((address_space(1)))

struct s {
        int i;
} __user;


extern void use0(void *);
extern void use1(void __user *);

void main(void)
{
        struct s s;
        int i;

        use0(&s);       // KO
        use0(&i);       // OK
        use1(&s);       // OK
        use1(&i);       // KO
}

/*
 * check-name: type-attribute-as
 *
 * check-error-start
type-attribute-as.c:16:15: warning: incorrect type in argument 1 (different addr
type-attribute-as.c:16:15:    expected void *<noident>
type-attribute-as.c:16:15:    got struct s <asn:1>*<noident>
type-attribute-as.c:19:15: warning: incorrect type in argument 1 (different addr
type-attribute-as.c:19:15:    expected void <asn:1>*<noident>
type-attribute-as.c:19:15:    got int *<noident>
 * check-error-end
 */
```

```
**********************************************************
     277 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/type-attribute-mod.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __noderef       __attribute__((noderef))

struct s {
        int i;
} __noderef;


void main(void)
{
        struct s s;

        s.i = 0;
}

/*
 * check-name: type-attribute-mod
 *
 * check-error-start
type-attribute-mod.c:12:9: warning: dereference of noderef expression
 * check-error-end
 */
```

**********
*496 Fri Dec 21 15:01:16 2018*
*new/usr/src/tools/smatch/src/validation/type1.c*
*10063 basic support for smatch*
*10153 checkpaths shouldn't check packaging exceptions*
**********

```
/*
 * Sparse used to get this wrong.
 *
 * When evaluating the argument to the inline function for the array, Sparse
 * didn't properly demote the "char []" to a "char *", but instead it would
 * follow the dereference and get a "struct hello".
 *
 * Which made no sense at all.
 */

static inline int deref(const char *s)
{
        return *s;
}

struct hello {
        char array[10];
};

static int test(struct hello *arg)
{
        return deref(arg->array);
}

/*
 * check-name: "char []" to "char *" demotion
 */
```

```
**********************************************************
     288 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/typedef_shadow.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
typedef int T;
static void f(int T)
{
        static T a;
}
/*
 * check-name: typedef shadowing
 * check-error-start:
typedef_shadow.c:4:16: warning: 'T' has implicit type
typedef_shadow.c:4:18: error: Expected ; at end of declaration
typedef_shadow.c:4:18: error: got a
 * check-error-end:
 */
```

```
**********************************************************
     393 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/typeof-addresspace.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __as            __attribute__((address_space(1)))

static void test_as(void)
{
        int __as obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;       /* check-should-pass */
        typeof(obj) *ptr4 = ptr;        /* check-should-pass */
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}

/*
 * check-name: typeof-addresspace.c
 * check-known-to-fail
 */
```

```
**********************************************************
     361 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/typeof-attribute.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __percpu __attribute__((noderef, address_space(3)))

/* Turn v back into a normal var. */
#define convert(v) \
       (*(typeof(v) __attribute__((address_space(0), force)) *)(&v))

int main(int argc, char *argv)
{
       unsigned int __percpu x;

       convert(x) = 0;
       return 0;
}
/*
 * check-name: Rusty Russell's typeof attribute casting.
 */
```

```
**********************************************************
    1901 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/typeof-mods.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __noderef       __attribute__((noderef))
#define __bitwise       __attribute__((bitwise))
#define __nocast        __attribute__((nocast))
#define __safe          __attribute__((safe))

static void test_spec(void)
{
        unsigned int obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}

static void test_const(void)
{
        const int obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        ptr = ptr;
        ptr = &obj;
}

static void test_volatile(void)
{
        volatile int obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}

static void test_bitwise(void)
{
        typedef int __bitwise type_t;
        type_t obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}
```



```
static void test_static(void)
{
        static int obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}

static void test_tls(void)
{
        __thread int obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}

static void test_nocast(void)
{
        int __nocast obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}

/*
 * check-name: typeof-mods
 *
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
     312 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/typeof-noderef.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __noderef       __attribute__((noderef))

static void test_noderef(void)
{
        int __noderef obj, *ptr;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        ptr = ptr;
        ptr = &obj;
}

/*
 * check-name: typeof-noderef
 * check-known-to-fail
 *
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
     373 Fri Dec 21 15:01:16 2018
new/usr/src/tools/smatch/src/validation/typeof-safe.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#define __safe          __attribute__((safe))

static void test_safe(void)
{
        int __safe obj, *ptr;
        typeof(obj) var = obj;
        typeof(ptr) ptr2 = ptr;
        typeof(*ptr) var2 = obj;
        typeof(*ptr) *ptr3 = ptr;
        typeof(obj) *ptr4 = ptr;
        obj = obj;
        ptr = ptr;
        ptr = &obj;
        obj = *ptr;
}

/*
 * check-name: typeof-safe
 * check-known-to-fail
 *
 * check-error-start
 * check-error-end
 */
```

```
**********************************************************
    1632 Fri Dec 21 15:01:17 2018
new/usr/src/tools/smatch/src/validation/typesign.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
static unsigned int * s_to_u_return(signed int *sp)
{
        return sp;
}

static signed int * u_to_s_return(unsigned int *up)
{
        return up;
}

static unsigned int * s_to_u_init(signed int *sp)
{
        unsigned int *up = sp;
        return up;
}

static signed int * u_to_s_init(unsigned int *up)
{
        signed int *sp = up;
        return sp;
}

static unsigned int * s_to_u_assign(signed int *sp)
{
        unsigned int *up;
        up = sp;
        return up;
}

static signed int * u_to_s_assign(unsigned int *up)
{
        signed int *sp;
        sp = up;
        return sp;
}

/*
 * check-name: -Wtypesign
 * check-command: sparse -Wtypesign $file
 *
 * check-error-start
typesign.c:3:16: warning: incorrect type in return expression (different signedn
typesign.c:3:16:    expected unsigned int *
typesign.c:3:16:    got signed int *sp
typesign.c:8:16: warning: incorrect type in return expression (different signedn
typesign.c:8:16:    expected signed int *
typesign.c:8:16:    got unsigned int *up
typesign.c:13:28: warning: incorrect type in initializer (different signedness)
typesign.c:13:28:    expected unsigned int *up
typesign.c:13:28:    got signed int *sp
typesign.c:19:26: warning: incorrect type in initializer (different signedness)
typesign.c:19:26:    expected signed int *sp
typesign.c:19:26:    got unsigned int *up
typesign.c:26:12: warning: incorrect type in assignment (different signedness)
typesign.c:26:12:    expected unsigned int *up
typesign.c:26:12:    got signed int *sp
typesign.c:33:12: warning: incorrect type in assignment (different signedness)
typesign.c:33:12:    expected signed int *sp
typesign.c:33:12:    got unsigned int *up
 * check-error-end
```



```
*/
```

```
**********************************************************
     165 Fri Dec 21 15:01:17 2018
new/usr/src/tools/smatch/src/validation/varargs1.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
1 extern int foo (const char *, ...);
2 static void sparse_error(const char err[])
3 {
4   foo("%s\n",err);
5 }
6 /*
7  * check-name: Varargs bogus warning regression test #1
8  */
```

```
**********************************************************
     189 Fri Dec 21 15:01:17 2018
new/usr/src/tools/smatch/src/validation/wide.c
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
   1 static char c = L'\x41';
   2 static int n = 1/(0x41 - L'\x41');
   3 /*
   4  * check-name: wide character constants
   5  *
   6  * check-error-start
   7 wide.c:2:17: warning: division by zero
   8  * check-error-end
   9  */
```

```
**********************************************************
    1378 Fri Dec 21 15:01:17 2018
new/usr/src/ucbcmd/rusage/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.

PROG= rusage

include ../Makefile.ucbcmd

INC1 = $(SRC)/ucbhead

FILEMODE=       755

LDLIBS +=  -L$(ROOT)/usr/ucblib -R/usr/ucblib -lucb
LDLIBS +=  -L$(ROOT)/usr/ucblib -R /usr/ucblib -lucb

CPPFLAGS = -I$(INC1) $(CPPFLAGS.master)
CERRWARN += -_gcc=-Wno-implicit-function-declaration

# missing prototypes
SMATCH=off

.KEEP_STATE:

all: $(PROG)

$(PROG):        $(PROG).c
        $(LINK.c) -o $@ $(PROG).c $(LDLIBS)
        $(POST_PROCESS)

install: all $(ROOTPROG)

clean:

lint:   lint_PROG

include ../Makefile.ucbtarg
```

```
**********************************************************
    1600 Fri Dec 21 15:01:17 2018
new/usr/src/ucbcmd/tset/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#
#
# Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
# Use is subject to license terms.
#
# Copyright (c) 2018, Joyent, Inc.

PROG= tset

include ../Makefile.ucbcmd

FILEMODE=       755

LDLIBS =  -L$(ROOT)/usr/ucblib -R/usr/ucblib $(LDLIBS.cmd) -lucb -ltermcap
LDLIBS =  -L$(ROOT)/usr/ucblib -R /usr/ucblib $(LDLIBS.cmd) -lucb -ltermcap

CPPFLAGS = -I$(SRC)/ucbhead $(CPPFLAGS.master)
CERRWARN += -_gcc=-Wno-implicit-function-declaration
CERRWARN += -_gcc=-Wno-parentheses
CERRWARN += -_gcc=-Wno-unused-variable
CERRWARN += -_gcc=-Wno-uninitialized

# missing prototypes
SMATCH=off

ROOTSYMLINK= $(ROOTBIN)/reset

.KEEP_STATE:

all: $(PROG)

$(PROG):        $(PROG).c
        $(LINK.c) -o $@ $(PROG).c $(LDLIBS)
        $(POST_PROCESS)

install: all $(ROOTBIN) $(ROOTPROG) $(ROOTSYMLINK)

$(ROOTSYMLINK):
        $(RM) $@
        $(SYMLINK) $(PROG) $@
```



```
clean:

lint:   lint_PROG

include ../Makefile.ucbtarg
```

```
**********************************************************
    1412 Fri Dec 21 15:01:17 2018
new/usr/src/ucbcmd/vipw/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
   1 #
   2 # CDDL HEADER START
   3 #
   4 # The contents of this file are subject to the terms of the
   5 # Common Development and Distribution License (the "License").
   6 # You may not use this file except in compliance with the License.
   7 #
   8 # You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9 # or http://www.opensolaris.org/os/licensing.
  10 # See the License for the specific language governing permissions
  11 # and limitations under the License.
  12 #
  13 # When distributing Covered Code, include this CDDL HEADER in each
  14 # file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15 # If applicable, add the following below this CDDL HEADER, with the
  16 # fields enclosed by brackets "[]" replaced with your own identifying
  17 # information: Portions Copyright [yyyy] [name of copyright owner]
  18 #
  19 # CDDL HEADER END
  20 #
  21 #
  22 # Copyright 2009 Sun Microsystems, Inc.  All rights reserved.
  23 # Use is subject to license terms.
  24 #
  25 # Copyright (c) 2018, Joyent, Inc.

  27 PROG= vipw

  29 include ../Makefile.ucbcmd

  31 INC1 = $(SRC)/ucbhead

  33 LDLIBS =  -L$(ROOT)/usr/ucblib -R/usr/ucblib $(LDLIBS.cmd) -lucb
  32 LDLIBS =  -L$(ROOT)/usr/ucblib -R /usr/ucblib $(LDLIBS.cmd) -lucb

  35 CPPFLAGS = -I$(INC1) $(CPPFLAGS.master)
  36 CERRWARN += -_gcc=-Wno-implicit-function-declaration
  37 CERRWARN += -_gcc=-Wno-unused-variable

  39 # missing prototypes
  40 SMATCH=off

  42 .KEEP_STATE:

  44 all: $(PROG)

  46 $(PROG):        $(PROG).c
  47         $(LINK.c) -o $@ $(PROG).c $(LDLIBS)
  48         $(POST_PROCESS)

  50 install: all $(ROOTPROG)

  52 clean:

  54 lint:   lint_PROG

  56 include ../Makefile.ucbtarg
```

```
**********************************************************
   21461 Fri Dec 21 15:01:17 2018
new/usr/src/uts/Makefile.uts
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# CDDL HEADER START
#
# The contents of this file are subject to the terms of the
# Common Development and Distribution License (the "License").
# You may not use this file except in compliance with the License.
#
# You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
# or http://www.opensolaris.org/os/licensing.
# See the License for the specific language governing permissions
# and limitations under the License.
#
# When distributing Covered Code, include this CDDL HEADER in each
# file and include the License file at usr/src/OPENSOLARIS.LICENSE.
# If applicable, add the following below this CDDL HEADER, with the
# fields enclosed by brackets "[]" replaced with your own identifying
# information: Portions Copyright [yyyy] [name of copyright owner]
#
# CDDL HEADER END
#

#
# Copyright (c) 1991, 2010, Oracle and/or its affiliates. All rights reserved.
# Copyright (c) 2011 Bayard G. Bell. All rights reserved.
# Copyright (c) 2011 by Delphix. All rights reserved.
# Copyright (c) 2013 Andrew Stormont.  All rights reserved.
# Copyright 2016 Hans Rosenfeld <rosenfeld@grumpf.hope-2000.org>
# Copyright (c) 2018, Joyent, Inc.
#

#
#       This Makefile contains the common targets and definitions for
#       all kernels. It is to be included in the Makefiles for specific
#       implementation architectures and processor architecture dependent
#       modules: i.e.: all driving kernel Makefiles.
#
#       Include global definitions:
#
include $(SRC)/Makefile.master

#
#       No text domain in the kernel.
#
DTEXTDOM =

#
#       Keep references to $(SRC)/common relative.
COMMONBASE=     $(UTSBASE)/../common

#
#       Setup build-specific vars
#       To add a build type:
#               add name to ALL_BUILDS32 & ALL_BUILDS64
#               set CLASS_name and OBJ_DIR_name
#               add targets to Makefile.targ
#

#
#       DEF_BUILDS is for def, lint, sischeck, and install
#       ALL_BUILDS is for everything else (all, clean, ...)
```



```
#
# The NOT_RELEASE_BUILD noise is to maintain compatibility with the
# gatekeeper's nightly build script.
#
DEF_BUILDS32                            = obj32
DEF_BUILDS64                            = obj64
DEF_BUILDSONLY64                        = obj64
$(NOT_RELEASE_BUILD)DEF_BUILDS32        = debug32
$(NOT_RELEASE_BUILD)DEF_BUILDS64        = debug64
$(NOT_RELEASE_BUILD)DEF_BUILDSONLY64    = debug64
ALL_BUILDS32                            = obj32 debug32
ALL_BUILDS64                            = obj64 debug64
ALL_BUILDSONLY64                        = obj64 debug64

#
#       For modules in 64b dirs that aren't built 64b
#       or modules in 64b dirs that aren't built 32b we
#       need to create empty modlintlib files so global lint works
#
LINT32_BUILDS   = debug32
LINT64_BUILDS   = debug64

#
#       Build class (32b or 64b)
#
CLASS_OBJ32     = 32
CLASS_DBG32     = 32
CLASS_OBJ64     = 64
CLASS_DBG64     = 64
CLASS           = $(CLASS_$(BUILD_TYPE))

#
#       Build subdirectory
#
OBJS_DIR_OBJ32  = obj32
OBJS_DIR_DBG32  = debug32
OBJS_DIR_OBJ64  = obj64
OBJS_DIR_DBG64  = debug64
OBJS_DIR        = $(OBJS_DIR_$(BUILD_TYPE))

#
#       Create defaults so empty rules don't
#       confuse make
#
CLASS_          = 64
OBJS_DIR_       = debug64

#
#       Build tools
#
CC_sparc_32     = $(sparc_CC)
CC_sparc_64     = $(sparcv9_CC)

CC_i386_32      = $(i386_CC)
CC_i386_64      = $(amd64_CC)
CC_amd64_64     = $(amd64_CC)

CC              = $(CC_$(MACH)_$(CLASS))

AS_sparc_32     = $(sparc_AS)
AS_sparc_64     = $(sparcv9_AS)

AS_i386_32      = $(i386_AS)
AS_i386_64      = $(amd64_AS)
AS_amd64_64     = $(amd64_AS)
```

```
127 AS              = $(AS_$(MACH)_$(CLASS))

129 LD_sparc_32     = $(sparc_LD)
130 LD_sparc_64     = $(sparcv9_LD)

132 LD_i386_32      = $(i386_LD)
133 LD_i386_64      = $(amd64_LD)
134 LD_amd64_64     = $(amd64_LD)

136 LD              = $(LD_$(MACH)_$(CLASS))

138 LINT_sparc_32   = $(sparc_LINT)
139 LINT_sparc_64   = $(sparcv9_LINT)

141 LINT_i386_32    = $(i386_LINT)
142 LINT_i386_64    = $(amd64_LINT)
143 LINT_amd64_64   = $(amd64_LINT)

145 LINT            = $(LINT_$(MACH)_$(CLASS))

147 MODEL_32        = ilp32
148 MODEL_64        = lp64
149 MODEL           = $(MODEL_$(CLASS))

151 #
152 #       Build rules for linting the kernel.
153 #
154 LHEAD = $(ECHO) "\n$@";

156 # Note: egrep returns "failure" if there are no matches, which is
157 # exactly the opposite of what we need.
158 LGREP.2 =       if egrep -v ' (_init|_fini|_info) '; then false; else true; fi

160 LTAIL =

162 LINT.c =        $(LINT) -c -dirout=$(LINTS_DIR) $(LINTFLAGS) $(LINT_DEFS) $(CPPF

164 # Please do not add new erroff directives here.  If you need to disable
165 # lint warnings in your module for things that cannot be fixed in any
166 # reasonable manner, please augment LINTTAGS in your module Makefile
167 # instead.
168 LINTTAGS        = -erroff=E_INCONS_ARG_DECL2
169 LINTTAGS        += -erroff=E_INCONS_VAL_TYPE_DECL2

171 LINTFLAGS_sparc_32      = $(LINTCCMODE) -nsxmuF -errtags=yes
172 LINTFLAGS_sparc_64      = $(LINTFLAGS_sparc_32) -m64
173 LINTFLAGS_i386_32       = $(LINTCCMODE) -nsxmuF -errtags=yes
174 LINTFLAGS_i386_64       = $(LINTFLAGS_i386_32) -m64

176 LINTFLAGS       = $(LINTFLAGS_$(MACH)_$(CLASS)) $(LINTTAGS)
177 LINTFLAGS       += $(C99LMODE)

179 #
180 #       Override this variable to modify the name of the lint target.
181 #
182 LINT_MODULE=    $(MODULE)

184 #
185 #       Build the compile/assemble lines:
186 #
187 EXTRA_OPTIONS           =
188 AS_DEFS                 = -D_ASM -D__STDC__=0

190 ALWAYS_DEFS_32          = -D_KERNEL -D_SYSCALL32 -D_DDI_STRICT
191 ALWAYS_DEFS_64          = -D_KERNEL -D_SYSCALL32 -D_SYSCALL32_IMPL -D_ELF64 \
192                         -D_DDI_STRICT
```



```
193 #
194 # XX64  This should be defined by the compiler!
195 #
196 ALWAYS_DEFS_64          += -Dsun -D__sun -D__SVR4
197 ALWAYS_DEFS             = $(ALWAYS_DEFS_$(CLASS))

199 #
200 #       CPPFLAGS is deliberatly set with a "=" and not a "+=".  For the kernel
201 #       the header include path should not look for header files outside of
202 #       the kernel code.  This "=" removes the search path built in
203 #       Makefile.master inside CPPFLAGS.  Ditto for AS_CPPFLAGS.
204 #
205 CPPFLAGS        = $(ALWAYS_DEFS) $(ALL_DEFS) $(CONFIG_DEFS) \
206                     $(INCLUDE_PATH) $(EXTRA_OPTIONS)
207 ASFLAGS         += -P
208 AS_CPPFLAGS     = $(ALWAYS_DEFS) $(ALL_DEFS) $(CONFIG_DEFS) $(AS_DEFS) \
209                     $(AS_INC_PATH) $(EXTRA_OPTIONS)

211 #
212 #       Make it (relatively) easy to share compilation options between
213 #       all kernel implementations.
214 #

216 # Override the default, the kernel is squeaky clean
217 CERRWARN = -errtags=yes -errwarn=%all

219 CERRWARN += -_gcc=-Wno-missing-braces
220 CERRWARN += -_gcc=-Wno-sign-compare
221 CERRWARN += -_gcc=-Wno-unknown-pragmas
222 CERRWARN += -_gcc=-Wno-unused-parameter
223 CERRWARN += -_gcc=-Wno-missing-field-initializers

225 # DEBUG v. -nd make for frequent unused variables, empty conditions, etc. in
226 # -nd builds
227 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-unused
228 $(RELEASE_BUILD)CERRWARN += -_gcc=-Wno-empty-body

230 CERRWARN += -_smatch=-p=illumos_kernel
231 include $(SRC)/Makefile.smatch

233 #
234 # Unfortunately, _IOWR() is regularly used with a third argument of 0,
235 # so we have to disable all these smatch checks.
236 #
237 SMOFF += sizeof

239 CSTD = $(CSTD_GNU99)

241 CFLAGS_uts              =
242 CFLAGS_uts              += $(STAND_FLAGS_$(CLASS))
243 CFLAGS_uts              += $(CCVERBOSE)
244 CFLAGS_uts              += $(ILDOFF)
245 CFLAGS_uts              += $(XAOPT)
246 CFLAGS_uts              += $(CTF_FLAGS_$(CLASS))
247 CFLAGS_uts              += $(CERRWARN)
248 CFLAGS_uts              += $(CCNOAUTOINLINE)
249 CFLAGS_uts              += $(CCNOREORDER)
250 CFLAGS_uts              += $(CGLOBALSTATIC)
251 CFLAGS_uts              += $(EXTRA_CFLAGS)
252 CFLAGS_uts              += $(CSOURCEDEBUGFLAGS)
253 CFLAGS_uts              += $(CUSERFLAGS)

255 #
256 #       Declare that $(OBJECTS) and $(LINTS) can be compiled in parallel.
257 #       The DUMMY target is for those instances where OBJECTS and LINTS
258 #       are empty (to avoid an unconditional .PARALLEL).
```

```
.PARALLEL:      $(OBJECTS) $(LINTS) DUMMY

#
#       Expanded dependencies
#
DEF_DEPS        = $(DEF_BUILDS:%=def.%)
ALL_DEPS        = $(ALL_BUILDS:%=all.%)
CLEAN_DEPS      = $(ALL_BUILDS:%=clean.%)
CLOBBER_DEPS    = $(ALL_BUILDS:%=clobber.%)
LINT_DEPS       = $(DEF_BUILDS:%=lint.%)
MODLINTLIB_DEPS = $(DEF_BUILDS:%=modlintlib.%)
MODLIST_DEPS    = $(DEF_BUILDS:%=modlist.%)
CLEAN_LINT_DEPS = $(ALL_BUILDS:%=clean.lint.%)
INSTALL_DEPS    = $(DEF_BUILDS:%=install.%)
SYM_DEPS        = $(SYM_BUILDS:%=symcheck.%)
SISCHECK_DEPS   = $(DEF_BUILDS:%=sischeck.%)
SISCLEAN_DEPS   = $(ALL_BUILDS:%=sisclean.%)

#
#       Default module name
#
BINARY          = $(OBJS_DIR)/$(MODULE)

#
#       Default cleanup definitions
#
CLEANLINTFILES  = $(LINTS) $(MOD_LINT_LIB)
CLEANFILES      = $(OBJECTS) $(CLEANLINTFILES)
CLOBBERFILES    = $(BINARY) $(CLEANFILES)

#
#       Installation constants:
#
#               FILEMODE is the mode given to the kernel modules
#               CFILEMODE is the mode given to the '.conf' files
#
FILEMODE         = 755
DIRMODE          = 755
CFILEMODE        = 644

#
#       Special Installation Macros for the installation of '.conf' files.
#
#       These are unique because they are not installed from the current
#       working directory.
#
# Sigh.  Apparently at some time in the past there was a confusion on
# whether the name is SRC_CONFFILE or SRC_CONFILE.  Consistency with the
# other names would indicate SRC_CONFFILE, but the voting is >180 Makefiles
# with SRC_CONFILE and about 11 with SRC_CONFFILE.  Software development
# isn't a popularity contest, though, and so my inclination is to define
# both names for now and incrementally convert to SRC_CONFFILE to be consistent
# with the other names.
#
CONFFILE                = $(MODULE).conf
SRC_CONFFILE            = $(CONF_SRCDIR)/$(CONFFILE)
SRC_CONFILE             = $(SRC_CONFFILE)
ROOT_CONFFILE_32        = $(ROOTMODULE).conf
ROOT_CONFFILE_64        = $(ROOTMODULE:%/$(SUBDIR64)/$(MODULE)=%/$(MODULE)).conf
ROOT_CONFFILE           = $(ROOT_CONFFILE_$(CLASS))


INS.conffile= \
        $(RM) $@; $(INS) -s -m $(CFILEMODE) -f $(@D) $(SRC_CONFFILE)

#
```



```
# The CTF merge of child kernel modules is performed against one of the genunix
# modules.  For Intel builds, all modules will be used with a single genunix:
# the one built in intel/genunix.  For SPARC builds, a given
# module may be
# used with one of a number of genunix files, depending on what platform the
# module is deployed on.  We merge against the sun4u genunix to optimize for
# the common case.  We also merge against the ip driver since networking is
# typically loaded and types defined therein are shared between many modules.
#
CTFMERGE_GUDIR_sparc    = sun4u
CTFMERGE_GUDIR_i386     = intel
CTFMERGE_GUDIR          = $(CTFMERGE_GUDIR_$(MACH))

CTFMERGE_GENUNIX        = \
        $(UTSBASE)/$(CTFMERGE_GUDIR)/genunix/$(OBJS_DIR)/genunix

#
# Used to uniquify a non-genunix module against genunix. $VERSION is used
# for the label.
#
# For the ease of developers dropping modules onto possibly unrelated systems,
# you can set NO_GENUNIX_UNIQUIFY= in the environment to skip uniquifying
# against genunix.
#
NO_GENUNIX_UNIQUIFY=$(POUND_SIGN)
CTFMERGE_GENUNIX_DFLAG=-d $(CTFMERGE_GENUNIX)
$(NO_GENUNIX_UNIQUIFY)CTF_GENUNIX_DFLAG=

CTFMERGE_UNIQUIFY_AGAINST_GENUNIX       = \
        $(CTFMERGE) $(CTFMRGFLAGS) -L VERSION \
        $(CTFMERGE_GENUNIX_DFLAG) -o $@ $(OBJECTS) $(CTFEXTRAOBJS)

#
# Used to merge the genunix module.
#
CTFMERGE_GENUNIX_MERGE          = \
        $(CTFMERGE) $(CTFMRGFLAGS) -L VERSION -o $@ \
        $(OBJECTS) $(CTFEXTRAOBJS) $(IPCTF_TARGET)

#
# We ctfmerge the ip objects into genunix to maximize the number of common types
# found there, thus maximizing the effectiveness of uniquification.  We don't
# want the genunix build to have to know about the individual ip objects, so we
# put them in an archive.  The genunix ctfmerge then includes this archive.
#
IPCTF           = $(IPDRV_DIR)/$(OBJS_DIR)/ipctf.a

#
# Rule for building fake shared libraries used for symbol resolution
# when building other modules.  -znoreloc is needed here to avoid
# tripping over code that isn't really suitable for shared libraries.
#
BUILD.SO                = \
        $(LD) -o $@ $(GSHARED) $(ZNORELOC) -h $(SONAME)

#
# SONAME defaults for common fake shared libraries.
#
$(LIBGEN)               := SONAME = $(MODULE)
$(PLATLIB)              := SONAME = misc/platmod
$(CPULIB)               := SONAME = 'cpu/$$CPU'
$(DTRACESTUBS)          := SONAME = dtracestubs

#
#       Installation directories
#
```

```
392 #
393 #       For now, 64b modules install into a subdirectory
394 #       of their 32b brethren.
395 #
396 SUBDIR64_sparc          = sparcv9
397 SUBDIR64_i386           = amd64
398 SUBDIR64                = $(SUBDIR64_$(MACH))

400 ROOT_MOD_DIR            = $(ROOT)/kernel

402 ROOT_KERN_DIR_32        = $(ROOT_MOD_DIR)
403 ROOT_BRAND_DIR_32       = $(ROOT_MOD_DIR)/brand
404 ROOT_DRV_DIR_32         = $(ROOT_MOD_DIR)/drv
405 ROOT_DTRACE_DIR_32      = $(ROOT_MOD_DIR)/dtrace
406 ROOT_EXEC_DIR_32        = $(ROOT_MOD_DIR)/exec
407 ROOT_FS_DIR_32          = $(ROOT_MOD_DIR)/fs
408 ROOT_SCHED_DIR_32       = $(ROOT_MOD_DIR)/sched
409 ROOT_SOCK_DIR_32        = $(ROOT_MOD_DIR)/socketmod
410 ROOT_STRMOD_DIR_32      = $(ROOT_MOD_DIR)/strmod
411 ROOT_IPP_DIR_32         = $(ROOT_MOD_DIR)/ipp
412 ROOT_SYS_DIR_32         = $(ROOT_MOD_DIR)/sys
413 ROOT_MISC_DIR_32        = $(ROOT_MOD_DIR)/misc
414 ROOT_KGSS_DIR_32        = $(ROOT_MOD_DIR)/misc/kgss
415 ROOT_SCSI_VHCI_DIR_32   = $(ROOT_MOD_DIR)/misc/scsi_vhci
416 ROOT_PMCS_FW_DIR_32     = $(ROOT_MOD_DIR)/misc/pmcs
417 ROOT_QLC_FW_DIR_32      = $(ROOT_MOD_DIR)/misc/qlc
418 ROOT_EMLXS_FW_DIR_32    = $(ROOT_MOD_DIR)/misc/emlxs
419 ROOT_NLMISC_DIR_32      = $(ROOT_MOD_DIR)/misc
420 ROOT_MACH_DIR_32        = $(ROOT_MOD_DIR)/mach
421 ROOT_CPU_DIR_32         = $(ROOT_MOD_DIR)/cpu
422 ROOT_TOD_DIR_32         = $(ROOT_MOD_DIR)/tod
423 ROOT_FONT_DIR_32        = $(ROOT_MOD_DIR)/fonts
424 ROOT_DACF_DIR_32        = $(ROOT_MOD_DIR)/dacf
425 ROOT_CRYPTO_DIR_32      = $(ROOT_MOD_DIR)/crypto
426 ROOT_MAC_DIR_32         = $(ROOT_MOD_DIR)/mac
427 ROOT_KICONV_DIR_32      = $(ROOT_MOD_DIR)/kiconv

429 ROOT_KERN_DIR_64        = $(ROOT_MOD_DIR)/$(SUBDIR64)
430 ROOT_BRAND_DIR_64       = $(ROOT_MOD_DIR)/brand/$(SUBDIR64)
431 ROOT_DRV_DIR_64         = $(ROOT_MOD_DIR)/drv/$(SUBDIR64)
432 ROOT_DTRACE_DIR_64      = $(ROOT_MOD_DIR)/dtrace/$(SUBDIR64)
433 ROOT_EXEC_DIR_64        = $(ROOT_MOD_DIR)/exec/$(SUBDIR64)
434 ROOT_FS_DIR_64          = $(ROOT_MOD_DIR)/fs/$(SUBDIR64)
435 ROOT_SCHED_DIR_64       = $(ROOT_MOD_DIR)/sched/$(SUBDIR64)
436 ROOT_SOCK_DIR_64        = $(ROOT_MOD_DIR)/socketmod/$(SUBDIR64)
437 ROOT_STRMOD_DIR_64      = $(ROOT_MOD_DIR)/strmod/$(SUBDIR64)
438 ROOT_IPP_DIR_64         = $(ROOT_MOD_DIR)/ipp/$(SUBDIR64)
439 ROOT_SYS_DIR_64         = $(ROOT_MOD_DIR)/sys/$(SUBDIR64)
440 ROOT_MISC_DIR_64        = $(ROOT_MOD_DIR)/misc/$(SUBDIR64)
441 ROOT_KGSS_DIR_64        = $(ROOT_MOD_DIR)/misc/kgss/$(SUBDIR64)
442 ROOT_SCSI_VHCI_DIR_64   = $(ROOT_MOD_DIR)/misc/scsi_vhci/$(SUBDIR64)
443 ROOT_PMCS_FW_DIR_64     = $(ROOT_MOD_DIR)/misc/pmcs/$(SUBDIR64)
444 ROOT_QLC_FW_DIR_64      = $(ROOT_MOD_DIR)/misc/qlc/$(SUBDIR64)
445 ROOT_EMLXS_FW_DIR_64    = $(ROOT_MOD_DIR)/misc/emlxs/$(SUBDIR64)
446 ROOT_NLMISC_DIR_64      = $(ROOT_MOD_DIR)/misc/$(SUBDIR64)
447 ROOT_MACH_DIR_64        = $(ROOT_MOD_DIR)/mach/$(SUBDIR64)
448 ROOT_CPU_DIR_64         = $(ROOT_MOD_DIR)/cpu/$(SUBDIR64)
449 ROOT_TOD_DIR_64         = $(ROOT_MOD_DIR)/tod/$(SUBDIR64)
450 ROOT_FONT_DIR_64        = $(ROOT_MOD_DIR)/fonts/$(SUBDIR64)
451 ROOT_DACF_DIR_64        = $(ROOT_MOD_DIR)/dacf/$(SUBDIR64)
452 ROOT_CRYPTO_DIR_64      = $(ROOT_MOD_DIR)/crypto/$(SUBDIR64)
453 ROOT_MAC_DIR_64         = $(ROOT_MOD_DIR)/mac/$(SUBDIR64)
454 ROOT_KICONV_DIR_64      = $(ROOT_MOD_DIR)/kiconv/$(SUBDIR64)

456 ROOT_KERN_DIR           = $(ROOT_KERN_DIR_$(CLASS))
```



```
457 ROOT_BRAND_DIR          = $(ROOT_BRAND_DIR_$(CLASS))
458 ROOT_DRV_DIR            = $(ROOT_DRV_DIR_$(CLASS))
459 ROOT_DTRACE_DIR         = $(ROOT_DTRACE_DIR_$(CLASS))
460 ROOT_EXEC_DIR           = $(ROOT_EXEC_DIR_$(CLASS))
461 ROOT_FS_DIR             = $(ROOT_FS_DIR_$(CLASS))
462 ROOT_SCHED_DIR          = $(ROOT_SCHED_DIR_$(CLASS))
463 ROOT_SOCK_DIR           = $(ROOT_SOCK_DIR_$(CLASS))
464 ROOT_STRMOD_DIR         = $(ROOT_STRMOD_DIR_$(CLASS))
465 ROOT_IPP_DIR            = $(ROOT_IPP_DIR_$(CLASS))
466 ROOT_SYS_DIR            = $(ROOT_SYS_DIR_$(CLASS))
467 ROOT_MISC_DIR           = $(ROOT_MISC_DIR_$(CLASS))
468 ROOT_KGSS_DIR           = $(ROOT_KGSS_DIR_$(CLASS))
469 ROOT_SCSI_VHCI_DIR      = $(ROOT_SCSI_VHCI_DIR_$(CLASS))
470 ROOT_PMCS_FW_DIR        = $(ROOT_PMCS_FW_DIR_$(CLASS))
471 ROOT_QLC_FW_DIR         = $(ROOT_QLC_FW_DIR_$(CLASS))
472 ROOT_EMLXS_FW_DIR       = $(ROOT_EMLXS_FW_DIR_$(CLASS))
473 ROOT_NLMISC_DIR         = $(ROOT_NLMISC_DIR_$(CLASS))
474 ROOT_MACH_DIR           = $(ROOT_MACH_DIR_$(CLASS))
475 ROOT_CPU_DIR            = $(ROOT_CPU_DIR_$(CLASS))
476 ROOT_TOD_DIR            = $(ROOT_TOD_DIR_$(CLASS))
477 ROOT_FONT_DIR           = $(ROOT_FONT_DIR_$(CLASS))
478 ROOT_DACF_DIR           = $(ROOT_DACF_DIR_$(CLASS))
479 ROOT_CRYPTO_DIR         = $(ROOT_CRYPTO_DIR_$(CLASS))
480 ROOT_MAC_DIR            = $(ROOT_MAC_DIR_$(CLASS))
481 ROOT_KICONV_DIR         = $(ROOT_KICONV_DIR_$(CLASS))
482 ROOT_FIRMWARE_DIR       = $(ROOT_MOD_DIR)/firmware

484 ROOT_MOD_DIRS_32        = $(ROOT_BRAND_DIR_32) $(ROOT_DRV_DIR_32)
485 ROOT_MOD_DIRS_32        = $(ROOT_BRAND_DIR_32) $(ROOT_DRV_DIR_32)
486 ROOT_MOD_DIRS_32        += $(ROOT_EXEC_DIR_32) $(ROOT_DTRACE_DIR_32)
487 ROOT_MOD_DIRS_32        += $(ROOT_FS_DIR_32) $(ROOT_SCHED_DIR_32)
488 ROOT_MOD_DIRS_32        += $(ROOT_STRMOD_DIR_32) $(ROOT_SYS_DIR_32)
489 ROOT_MOD_DIRS_32        += $(ROOT_IPP_DIR_32) $(ROOT_SOCK_DIR_32)
490 ROOT_MOD_DIRS_32        += $(ROOT_MISC_DIR_32) $(ROOT_MACH_DIR_32)
491 ROOT_MOD_DIRS_32        += $(ROOT_KGSS_DIR_32)
492 ROOT_MOD_DIRS_32        += $(ROOT_SCSI_VHCI_DIR_32)
493 ROOT_MOD_DIRS_32        += $(ROOT_PMCS_FW_DIR_32)
494 ROOT_MOD_DIRS_32        += $(ROOT_QLC_FW_DIR_32)
495 ROOT_MOD_DIRS_32        += $(ROOT_EMLXS_FW_DIR_32)
496 ROOT_MOD_DIRS_32        += $(ROOT_CPU_DIR_32) $(ROOT_FONT_DIR_32)
497 ROOT_MOD_DIRS_32        += $(ROOT_TOD_DIR_32) $(ROOT_DACF_DIR_32)
498 ROOT_MOD_DIRS_32        += $(ROOT_CRYPTO_DIR_32) $(ROOT_MAC_DIR_32)
499 ROOT_MOD_DIRS_32        += $(ROOT_KICONV_DIR_32)
500 ROOT_MOD_DIRS_32        += $(ROOT_FIRMWARE_DIR)

502 USR_MOD_DIR            = $(ROOT)/usr/kernel

504 USR_DRV_DIR_32          = $(USR_MOD_DIR)/drv
505 USR_EXEC_DIR_32         = $(USR_MOD_DIR)/exec
506 USR_FS_DIR_32           = $(USR_MOD_DIR)/fs
507 USR_SCHED_DIR_32        = $(USR_MOD_DIR)/sched
508 USR_SOCK_DIR_32         = $(USR_MOD_DIR)/socketmod
509 USR_STRMOD_DIR_32       = $(USR_MOD_DIR)/strmod
510 USR_SYS_DIR_32          = $(USR_MOD_DIR)/sys
511 USR_MISC_DIR_32         = $(USR_MOD_DIR)/misc
512 USR_DACF_DIR_32         = $(USR_MOD_DIR)/dacf
513 USR_PCBE_DIR_32         = $(USR_MOD_DIR)/pcbe
514 USR_DTRACE_DIR_32       = $(USR_MOD_DIR)/dtrace
515 USR_BRAND_DIR_32        = $(USR_MOD_DIR)/brand

517 USR_DRV_DIR_64          = $(USR_MOD_DIR)/drv/$(SUBDIR64)
518 USR_EXEC_DIR_64         = $(USR_MOD_DIR)/exec/$(SUBDIR64)
519 USR_FS_DIR_64           = $(USR_MOD_DIR)/fs/$(SUBDIR64)
520 USR_SCHED_DIR_64        = $(USR_MOD_DIR)/sched/$(SUBDIR64)
521 USR_SOCK_DIR_64         = $(USR_MOD_DIR)/socketmod/$(SUBDIR64)
522 USR_STRMOD_DIR_64       = $(USR_MOD_DIR)/strmod/$(SUBDIR64)
```

```
523 USR_SYS_DIR_64          = $(USR_MOD_DIR)/sys/$(SUBDIR64)
524 USR_MISC_DIR_64         = $(USR_MOD_DIR)/misc/$(SUBDIR64)
525 USR_DACF_DIR_64         = $(USR_MOD_DIR)/dacf/$(SUBDIR64)
526 USR_PCBE_DIR_64         = $(USR_MOD_DIR)/pcbe/$(SUBDIR64)
527 USR_DTRACE_DIR_64       = $(USR_MOD_DIR)/dtrace/$(SUBDIR64)
528 USR_BRAND_DIR_64        = $(USR_MOD_DIR)/brand/$(SUBDIR64)

530 USR_DRV_DIR             = $(USR_DRV_DIR_$(CLASS))
531 USR_EXEC_DIR            = $(USR_EXEC_DIR_$(CLASS))
532 USR_FS_DIR              = $(USR_FS_DIR_$(CLASS))
533 USR_SCHED_DIR           = $(USR_SCHED_DIR_$(CLASS))
534 USR_SOCK_DIR            = $(USR_SOCK_DIR_$(CLASS))
535 USR_STRMOD_DIR          = $(USR_STRMOD_DIR_$(CLASS))
536 USR_SYS_DIR             = $(USR_SYS_DIR_$(CLASS))
537 USR_MISC_DIR            = $(USR_MISC_DIR_$(CLASS))
538 USR_DACF_DIR            = $(USR_DACF_DIR_$(CLASS))
539 USR_PCBE_DIR            = $(USR_PCBE_DIR_$(CLASS))
540 USR_DTRACE_DIR          = $(USR_DTRACE_DIR_$(CLASS))
541 USR_BRAND_DIR           = $(USR_BRAND_DIR_$(CLASS))

543 USR_MOD_DIRS_32         = $(USR_DRV_DIR_32) $(USR_EXEC_DIR_32)
544 USR_MOD_DIRS_32         += $(USR_FS_DIR_32) $(USR_SCHED_DIR_32)
545 USR_MOD_DIRS_32         += $(USR_STRMOD_DIR_32) $(USR_SYS_DIR_32)
546 USR_MOD_DIRS_32         += $(USR_MISC_DIR_32) $(USR_DACF_DIR_32)
547 USR_MOD_DIRS_32         += $(USR_PCBE_DIR_32)
548 USR_MOD_DIRS_32         += $(USR_DTRACE_DIR_32) $(USR_BRAND_DIR_32)
549 USR_MOD_DIRS_32         += $(USR_SOCK_DIR_32)

551 #
552 #
553 #
554 include $(SRC)/Makefile.psm

556 #
557 #       The "-r" on the remove may be considered temporary, but is required
558 #       while the replacement of the SUNW,SPARCstation-10,SX directory by
559 #       a symbolic link is being propagated.
560 #
561 INS.slink1= $(RM) -r $@; $(SYMLINK) $(PLATFORM) $@
562 INS.slink2= $(RM) -r $@; $(SYMLINK) ../$(PLATFORM)/$(@F) $@
563 INS.slink3= $(RM) -r $@; $(SYMLINK) $(IMPLEMENTED_PLATFORM) $@
564 INS.slink4= $(RM) -r $@; $(SYMLINK) ../$(PLATFORM)/include $@
565 INS.slink5= $(RM) -r $@; $(SYMLINK) ../$(PLATFORM)/sbin $@
566 INS.slink6= $(RM) -r $@; $(SYMLINK) ../../$(PLATFORM)/lib/$(MODULE) $@
567 INS.slink7= $(RM) -r $@; $(SYMLINK) ../../$(PLATFORM)/sbin/$(@F) $@

569 ROOT_PLAT_LINKS          = $(PLAT_LINKS:%=$(ROOT_PLAT_DIR)/%)
570 ROOT_PLAT_LINKS_2        = $(PLAT_LINKS_2:%=$(ROOT_PLAT_DIR)/%)
571 USR_PLAT_LINKS           = $(PLAT_LINKS:%=$(USR_PLAT_DIR)/%)
572 USR_PLAT_LINKS_2         = $(PLAT_LINKS_2:%=$(USR_PLAT_DIR)/%)

574 #
575 # Collection of all relevant, delivered kernel modules.
576 #
577 # Note that we insist on building genunix first, because everything else
578 # uniquifies against it.  When doing a 'make' from usr/src/uts/, we'll enter
579 # the platform directories first.  These will cd into the corresponding genunix
580 # directory and build it.  So genunix /shouldn't/ get rebuilt when we get to
581 # building all the kernel modules.  However, due to an as-yet-unexplained
582 # problem with dependencies, sometimes it does get rebuilt, which then messes
583 # up the other modules.  So we always force the issue here rather than try to
584 # build genunix in parallel with everything else.
585 #
586 PARALLEL_KMODS = $(DRV_KMODS) $(EXEC_KMODS) $(FS_KMODS) $(SCHED_KMODS) \
587                  $(TOD_KMODS) $(STRMOD_KMODS) $(SYS_KMODS) $(MISC_KMODS) \
588                  $(NLMISC_KMODS) $(MACH_KMODS) $(CPU_KMODS) $(GSS_KMODS) \
```



```
589                  $(MMU_KMODS) $(DACF_KMODS) $(EXPORT_KMODS) $(IPP_KMODS) \
590                  $(CRYPTO_KMODS) $(PCBE_KMODS) \
591                  $(DRV_KMODS_$(CLASS)) $(MISC_KMODS_$(CLASS)) $(MAC_KMODS) \
592                  $(BRAND_KMODS) $(KICONV_KMODS) \
593                  $(SOCKET_KMODS)

595 KMODS = $(GENUNIX_KMODS) $(PARALLEL_KMODS)

597 $(PARALLEL_KMODS): $(GENUNIX_KMODS)

599 LINT_KMODS = $(DRV_KMODS) $(EXEC_KMODS) $(FS_KMODS) $(SCHED_KMODS) \
600           $(TOD_KMODS) $(STRMOD_KMODS) $(SYS_KMODS) $(MISC_KMODS) \
601           $(MACH_KMODS) $(GSS_KMODS) $(DACF_KMODS) $(IPP_KMODS) \
602           $(CRYPTO_KMODS) $(PCBE_KMODS) \
603           $(DRV_KMODS_$(CLASS)) $(MISC_KMODS_$(CLASS)) $(MAC_KMODS) \
604           $(BRAND_KMODS) $(KICONV_KMODS) $(SOCKET_KMODS)

606 #
607 #       Files to be compiled with -xa, to generate basic block execution
608 #       count data.
609 #
610 #       There are several ways to compile parts of the kernel for kcov:
611 #               1)  Add targets to BB_FILES here or in other Makefiles
612 #                       (they must in the form of $(OBJS_DIR)/target.o)
613 #               2)  setenv BB_FILES '$(XXX_OBJS:%=$(OBJS_DIR)/%)'
614 #               3)  setenv BB_FILES '$(OBJECTS)'
615 #
616 #       Do NOT setenv CFLAGS -xa, as that will cause infinite recursion
617 #       in unix_bb.o
618 #
619 BB_FILES =
620 $(BB_FILES)     := XAOPT = -xa

622 #
623 #       The idea here is for unix_bb.o to be in all kernels except the
624 #       kernel which actually gets shipped to customers.  In practice,
625 #       $(RELEASE_BUILD) is on for a number of the late beta and fcs builds.
626 #
627 $(NOT_RELEASE_BUILD)$(OBJS_DIR)/unix_bb.o   := CPPFLAGS     += -DKCOV
628 $(NOT_RELEASE_BUILD)$(OBJS_DIR)/unix_bb.ln  := CPPFLAGS     += -DKCOV

630 #
631 #       Do not let unix_bb.o get compiled with -xa!
632 #
633 $(OBJS_DIR)/unix_bb.o   := XAOPT =

635 #
636 # Privilege files
637 #
638 PRIVS_AWK = $(SRC)/uts/common/os/privs.awk
639 PRIVS_DEF = $(SRC)/uts/common/os/priv_defs

641 #
642 # USB device data
643 #
644 USBDEVS_AWK =   $(SRC)/uts/common/io/usb/usbdevs2h.awk
645 USBDEVS_DATA =  $(SRC)/uts/common/io/usb/usbdevs
```

```
**********************************************************
    3702 Fri Dec 21 15:01:17 2018
new/usr/src/uts/intel/bnxe/Makefile
10063 basic support for smatch
10153 checkpaths shouldn't check packaging exceptions
**********************************************************
```

```
#
# This file and its contents are supplied under the terms of the
# Common Development and Distribution License ("CDDL"), version 1.0.
# You may only use this file in accordance with the terms of version
# 1.0 of the CDDL.
#
# A full copy of the text of the CDDL should have accompanied this
# source.  A copy of the CDDL is also available via the Internet at
# http://www.illumos.org/license/CDDL.
#

#
# Copyright (c) 2018, Joyent, Inc.
# Copyright (c) 2014, Joyent, Inc.  All rights reserved.
#

#
# uts/intel/bnxe/Makefile
#
#       This makefile drives the production of the bnxe
#       driver kernel module.
#
#       intel architecture dependent
#

#
#       Paths to the base of the uts directory trees
#
UTSBASE = ../..

#
#       Define the module and object file sets.
#
MODULE          = bnxe
OBJECTS         = $(BNXE_OBJS:%=$(OBJS_DIR)/%)
LINTS           = $(LINTS_DIR)/bnxe_lint.ln
ROOTMODULE      = $(ROOT_DRV_DIR)/$(MODULE)
SRCDIR          = $(UTSBASE)/common/io/bnxe
CONF_SRCDIR     = $(SRCDIR)

#
#       Include common rules.
#
include $(UTSBASE)/intel/Makefile.intel

#
#       Define targets
#
ALL_TARGET      = $(BINARY) $(CONFMOD)
LINT_TARGET     = $(MODULE).lint
INSTALL_TARGET  = $(BINARY) $(ROOTMODULE) $(ROOT_CONFFILE)

C99LMODE=       -Xc99=%all

#
#       Driver-specific flags
#       XXX inline bits were originally set to inline
#
CPPFLAGS        += -DLM_RXPKT_NON_CONTIGUOUS \
                   -DELINK_ENHANCEMENTS \
```



```
                   -DELINK_57711E_SUPPORT \
                   -DELINK_DEBUG \
                   -D__inline= \
                   -D_inline= \
                   -D__BASENAME__=\"bnxe\" \
                   -D__SunOS \
                   -D__S11 \
                   -DILLUMOS \
                   -DLITTLE_ENDIAN \
                   -DLITTLE_ENDIAN_HOST \
                   -D__LITTLE_ENDIAN \
                   -I$(SRCDIR)/577xx/include \
                   -I$(SRCDIR)/577xx/drivers/common/ecore \
                   -I$(SRCDIR)/577xx/drivers/common/include \
                   -I$(SRCDIR)/577xx/drivers/common/include/l4 \
                   -I$(SRCDIR)/577xx/drivers/common/include/l5 \
                   -I$(SRCDIR)/577xx/drivers/common/lm/device \
                   -I$(SRCDIR)/577xx/drivers/common/lm/fw \
                   -I$(SRCDIR)/577xx/drivers/common/lm/include \
                   -I$(SRCDIR)/577xx/drivers/common/lm/l4 \
                   -I$(SRCDIR)/577xx/drivers/common/lm/l4/include \
                   -I$(SRCDIR)/577xx/drivers/common/lm/l5 \
                   -I$(SRCDIR)/577xx/drivers/common/lm/l5/include \
                   -I$(SRCDIR)/577xx/hsi/hw/include \
                   -I$(SRCDIR)/577xx/hsi/mcp \
                   -I$(SRCDIR)

LDFLAGS         += -dy -r -Ndrv/ip -Nmisc/mac
#CERRWARN       += -_gcc=-Wno-old-style-declaration
CERRWARN        += -_gcc=-Wno-switch
CERRWARN        += -_gcc=-Wno-uninitialized
CERRWARN        += -_gcc=-Wno-parentheses
CERRWARN        += -_gcc=-Wno-unused-function
CERRWARN        += -_gcc=-Wno-unused-value
CERRWARN        += -_gcc=-Wno-unused-variable
CERRWARN        += -_gcc=-Wno-unused-but-set-variable
CERRWARN        += -_cc=-erroff=E_STATEMENT_NOT_REACHED
CERRWARN        += -_cc=-erroff=E_ARGUEMENT_MISMATCH
CERRWARN        += -_cc=-erroff=E_INTEGER_OVERFLOW_DETECTED
CERRWARN        += -_cc=-erroff=E_CONST_PROMOTED_UNSIGNED_LL
CERRWARN        += -_cc=-erroff=E_ENUM_VAL_OVERFLOWS_INT_MAX

# a whole mess
SMATCH=off

LINTTAGS        += -erroff=E_FUNC_RET_ALWAYS_IGNOR2
LINTTAGS        += -erroff=E_FUNC_RET_MAYBE_IGNORED2
LINTTAGS        += -erroff=E_STATIC_UNUSED
LINTTAGS        += -erroff=E_FUNC_SET_NOT_USED
LINTTAGS        += -erroff=E_BAD_PTR_CAST_ALIGN
LINTTAGS        += -erroff=E_CONSTANT_CONDITION
LINTTAGS        += -erroff=E_NOP_ELSE_STMT
LINTTAGS        += -erroff=E_NOP_IF_STMT
LINTTAGS        += -erroff=E_FUNC_ARG_UNUSED
LINTTAGS        += -erroff=E_FUNC_VAR_UNUSED
LINTTAGS        += -erroff=E_EXPR_NULL_EFFECT
LINTTAGS        += -erroff=E_STMT_NOT_REACHED
LINTTAGS        += -erroff=E_ASSIGN_NARROW_CONV
LINTTAGS        += -erroff=E_PTRDIFF_OVERFLOW
LINTTAGS        += -erroff=E_CASE_FALLTHRU
LINTTAGS        += -erroff=E_CONST_EXPR

#
#       Default build targets.
#
.KEEP_STATE:
```

```
126 def:            $(DEF_DEPS)

128 all:            $(ALL_DEPS)

130 clean:          $(CLEAN_DEPS)

132 clobber:        $(CLOBBER_DEPS)

134 lint:           $(LINT_DEPS)

136 modlintlib:     $(MODLINTLIB_DEPS)

138 clean.lint:     $(CLEAN_LINT_DEPS)

140 install:        $(INSTALL_DEPS)

142 #
143 #       Include common targets.
144 #
145 include $(UTSBASE)/intel/Makefile.targ
```